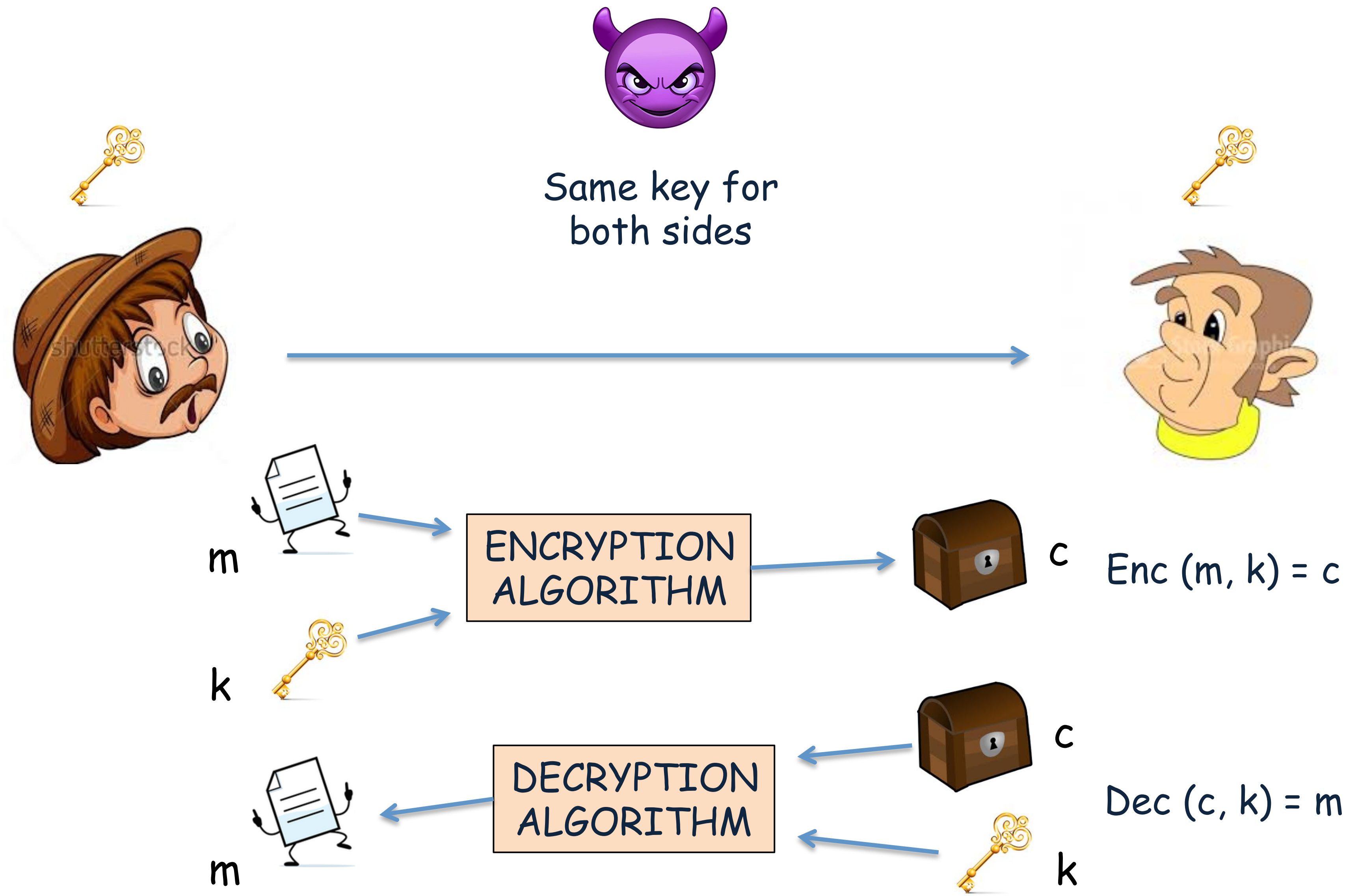
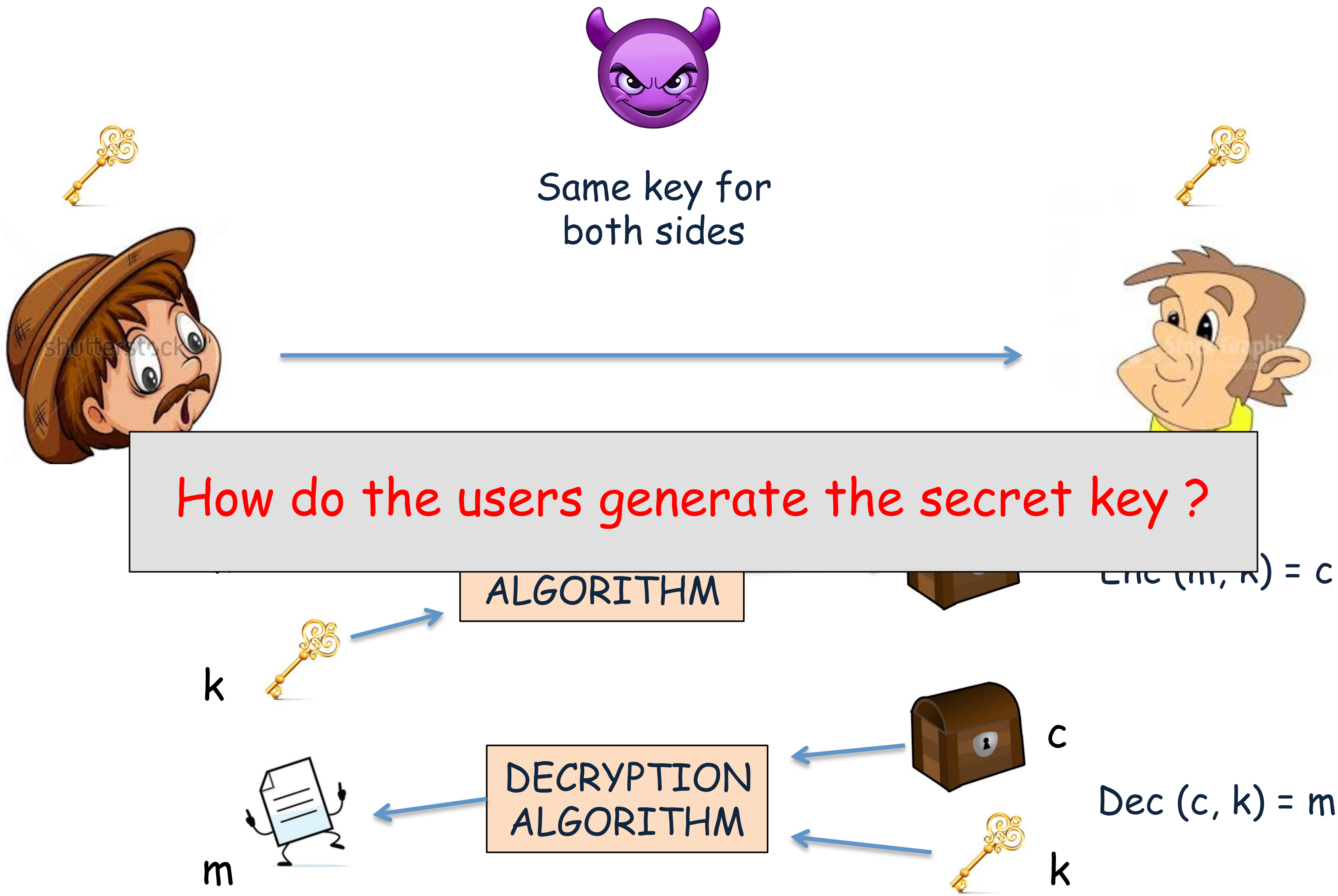
Public-Key Cryptography

Murat Osmanoglu

Symmetric Encryption



Symmetric Encryption



Key Exchange

A naïve approach

U_1

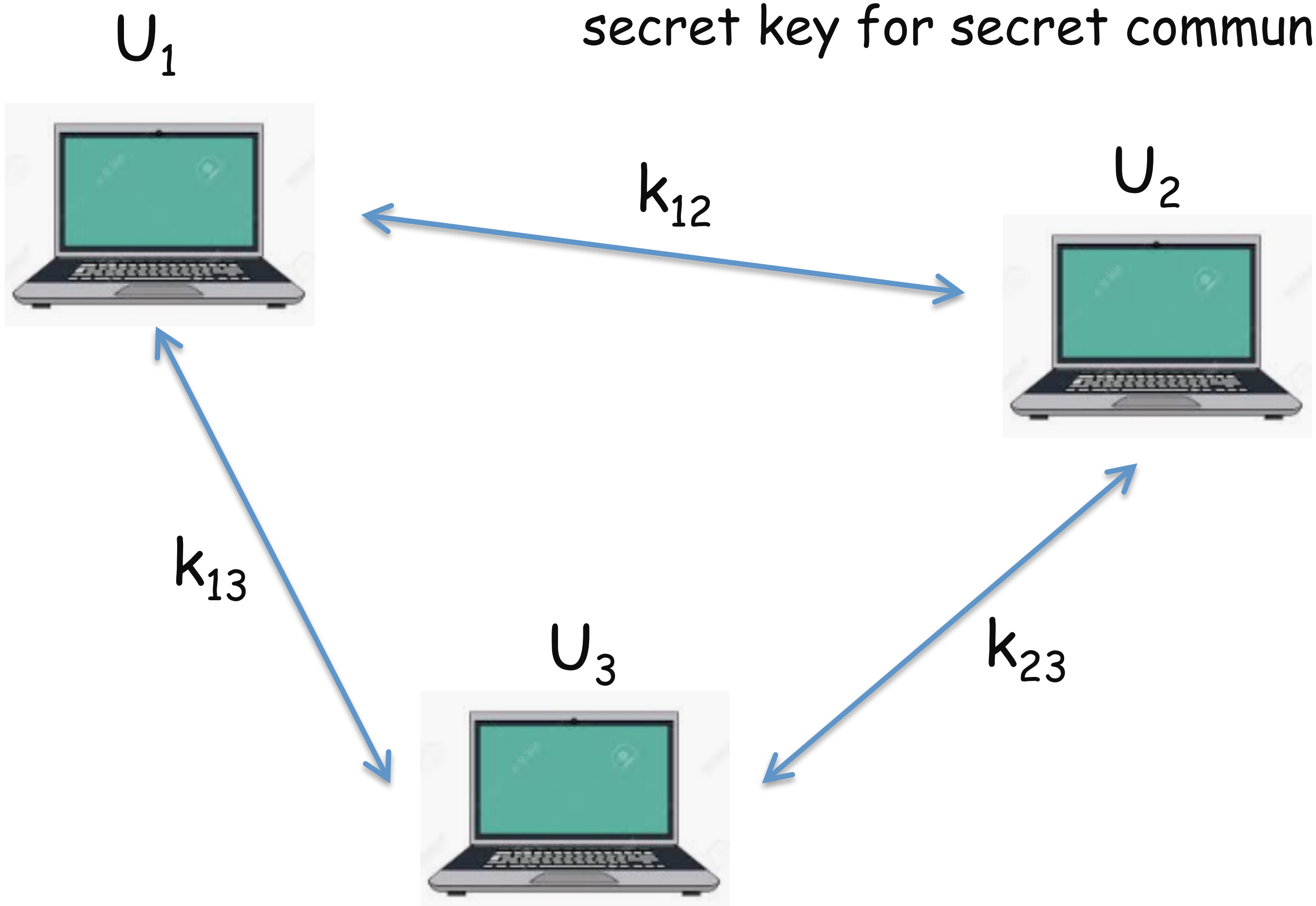
U_2

U_3

Key Exchange

A naïve approach

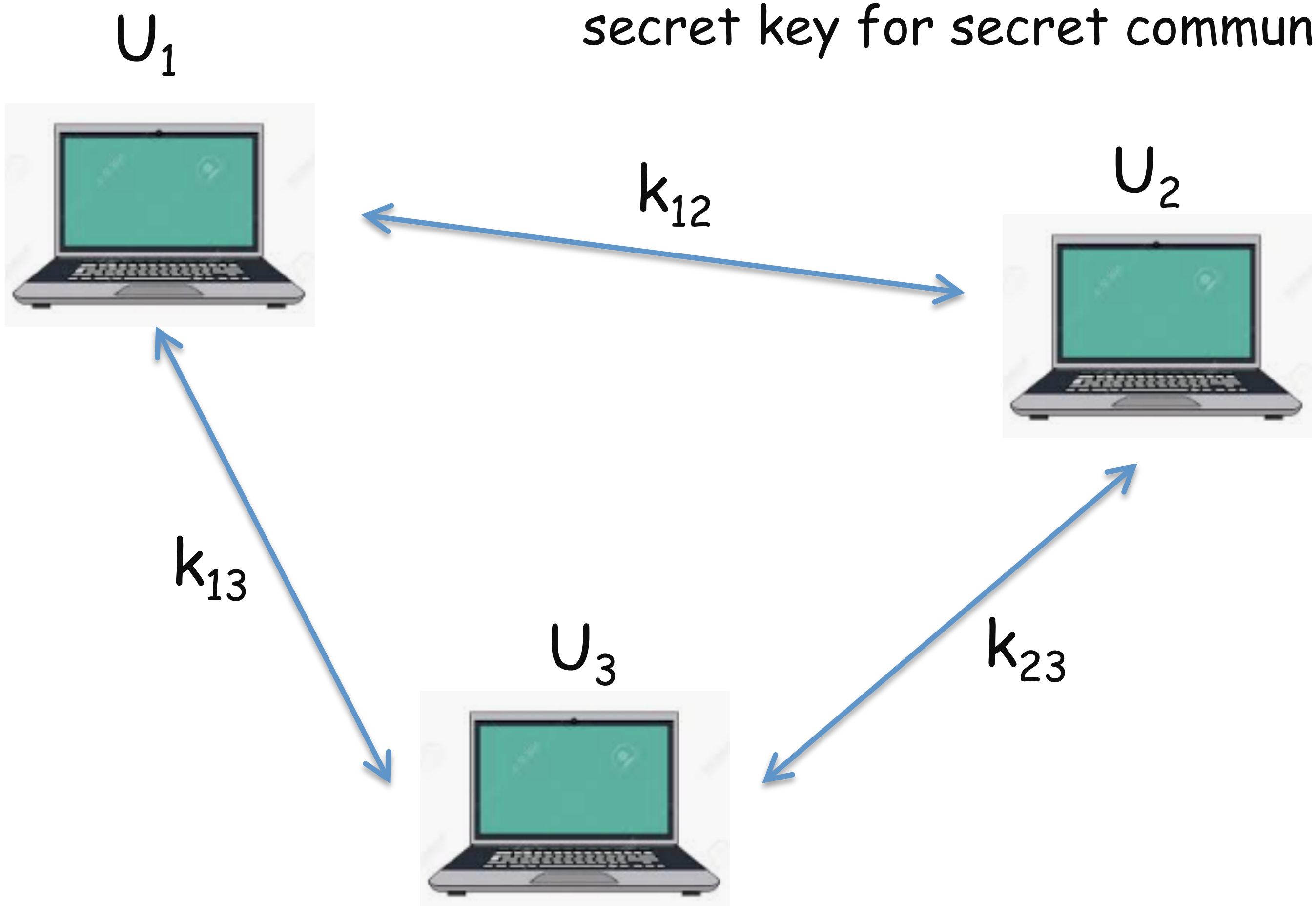
- each pair of users should share a secret key for secret communication



Key Exchange

A naïve approach

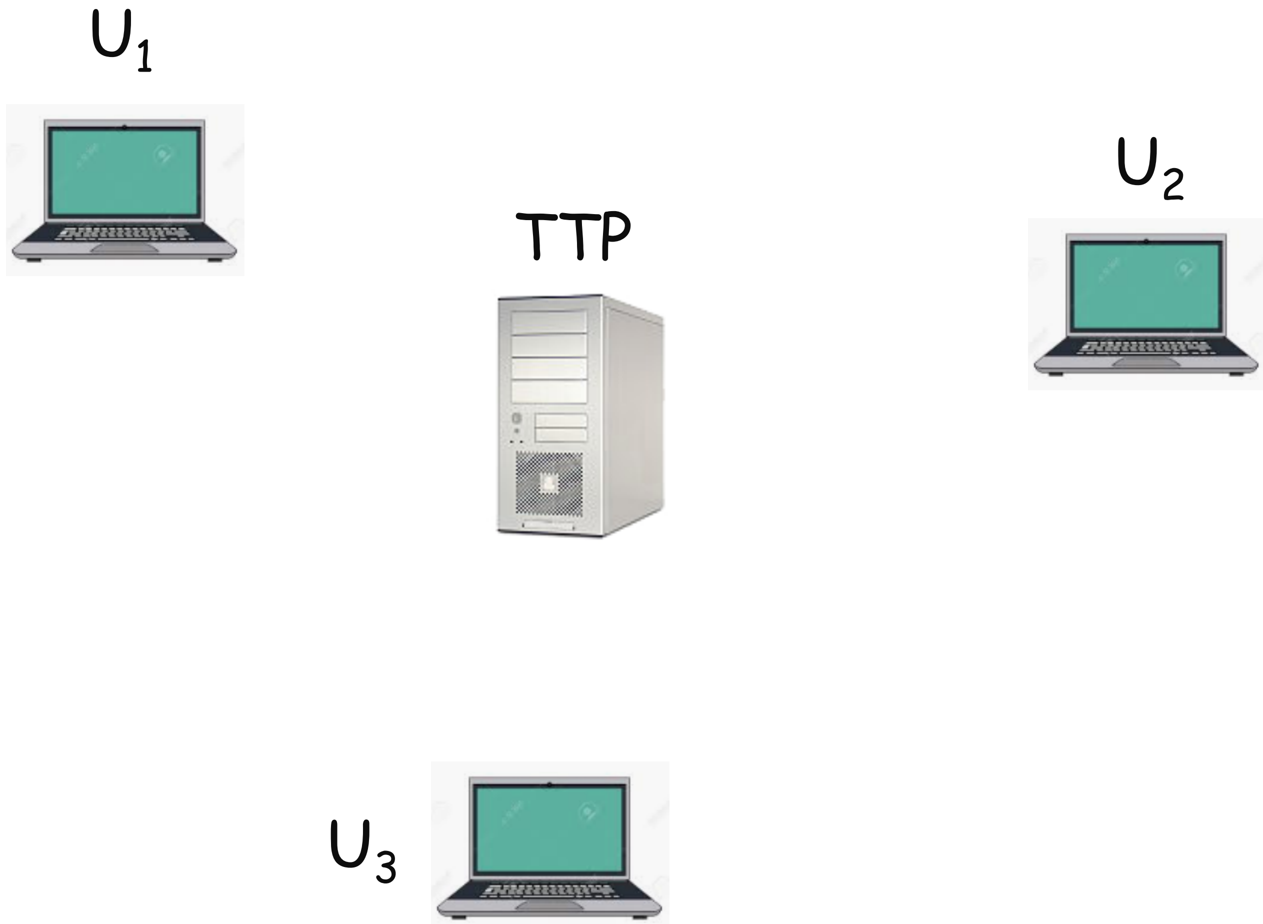
- each pair of users should share a secret key for secret communication



- each user should store O(n) secret keys

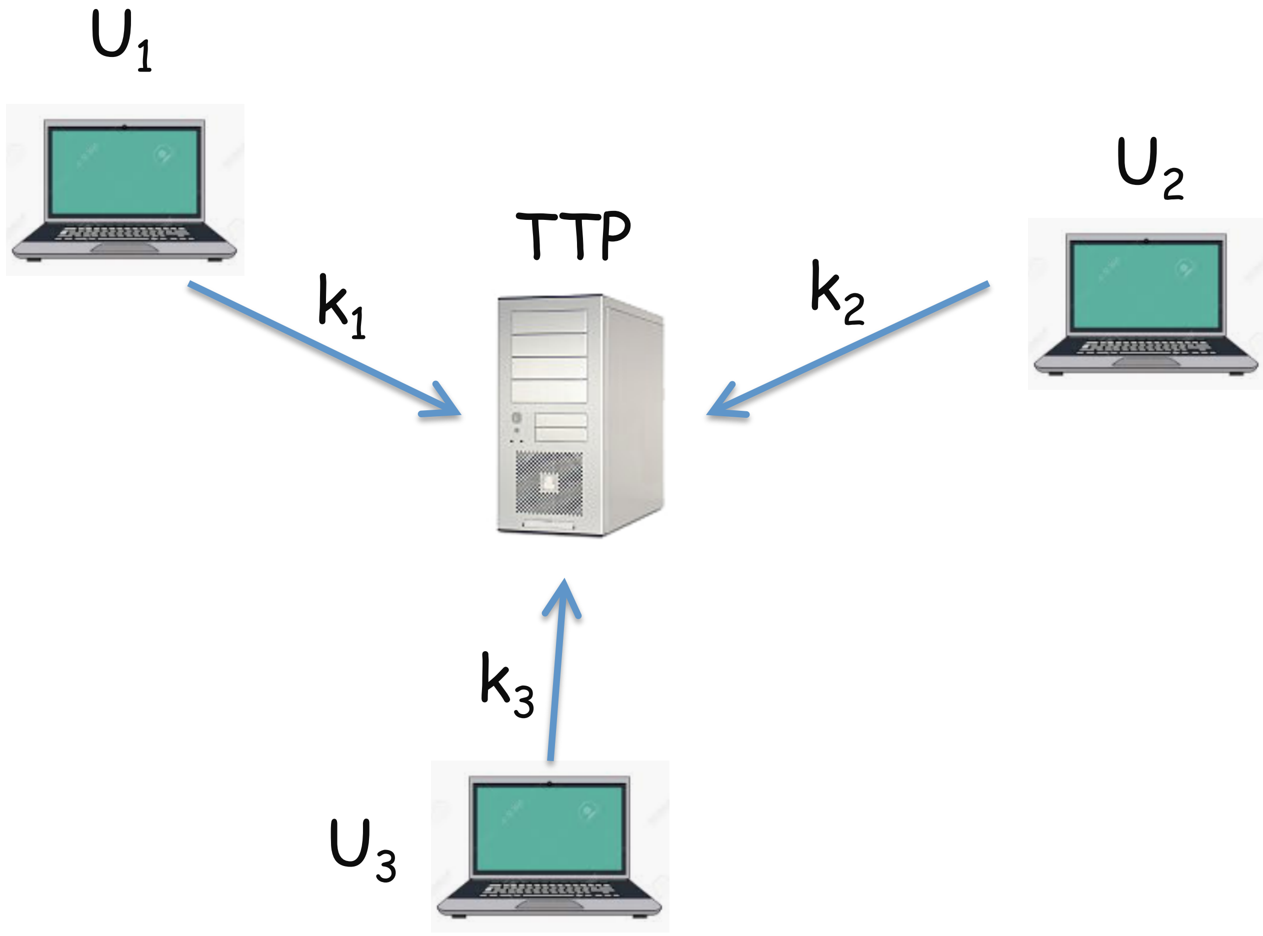
Key Exchange

Trusted Third Party



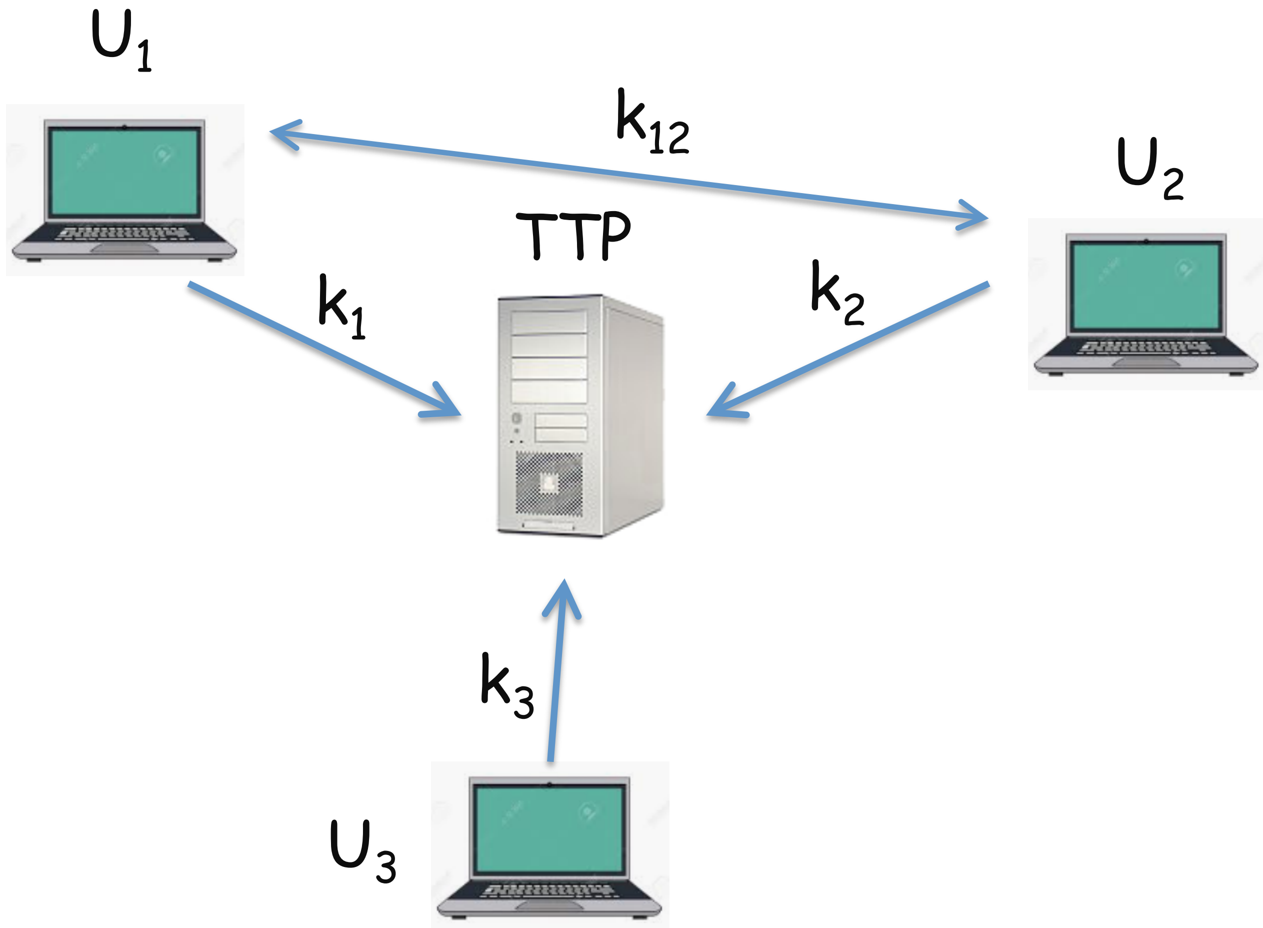
Key Exchange

Trusted Third Party



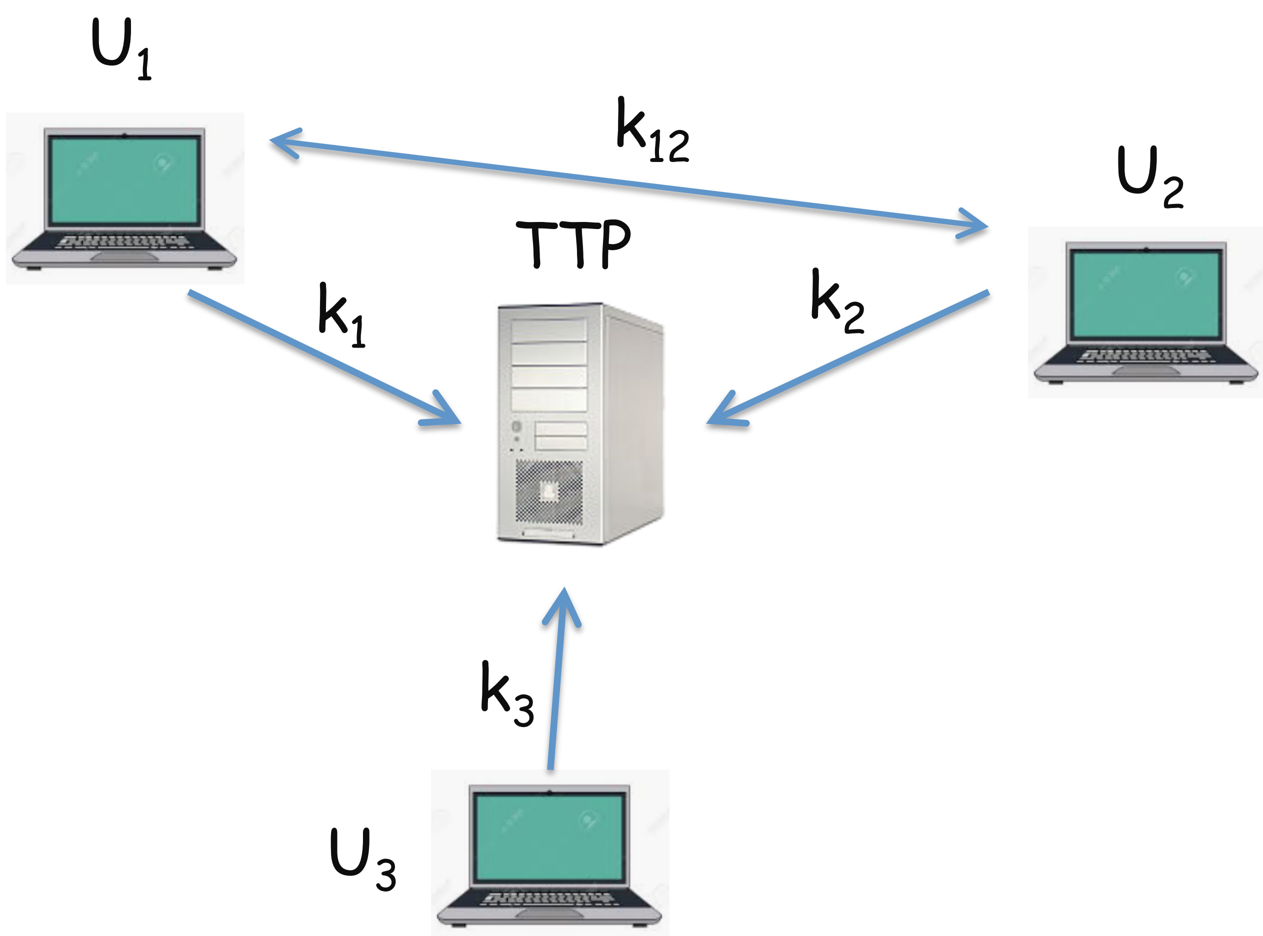
Key Exchange

Trusted Third Party



Key Exchange

Trusted Third Party



- How U_1 and U_2 generate the secret key k_{12} ?

Key Exchange

Trusted Third Party

U_1

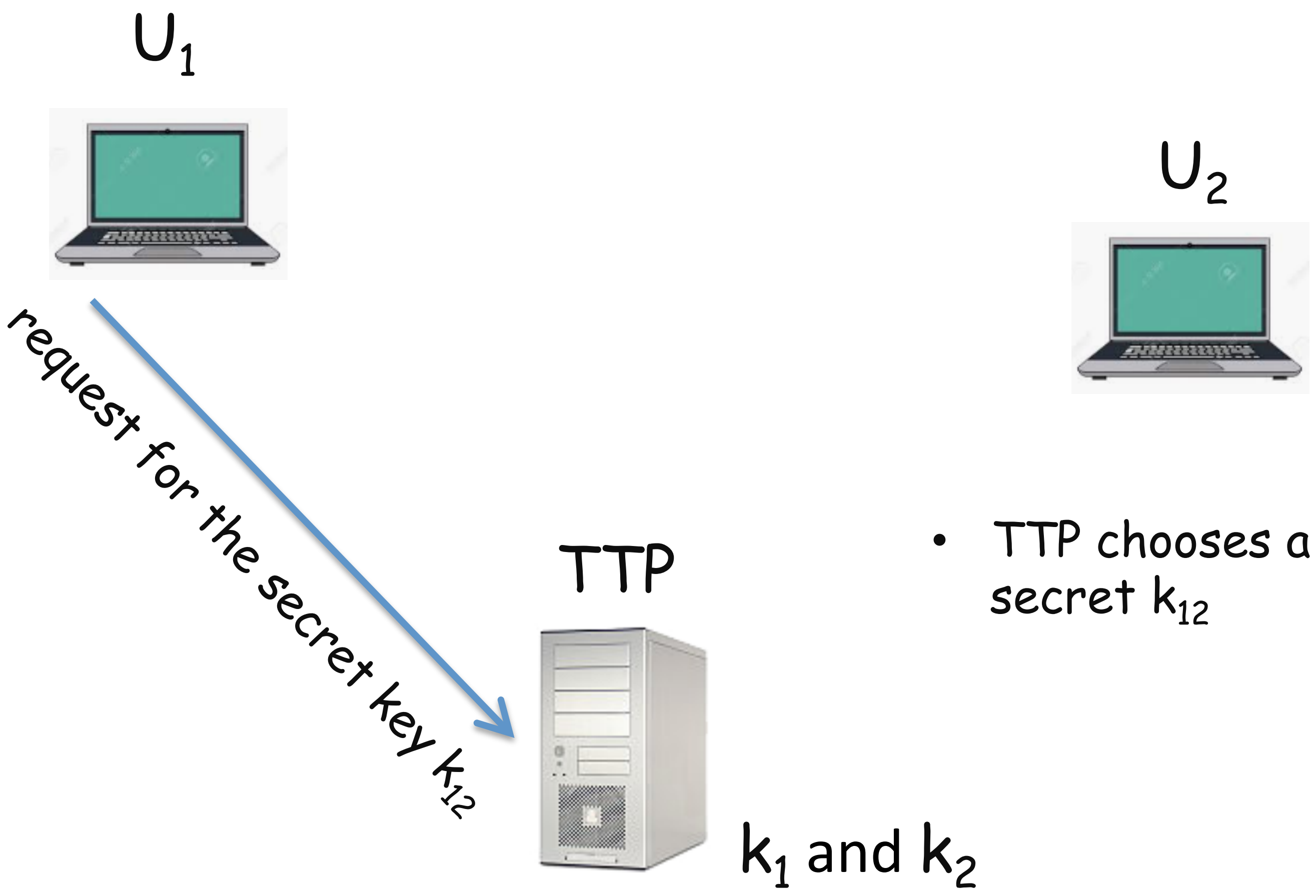
U_2

TTP

k_1 and k_2

Key Exchange

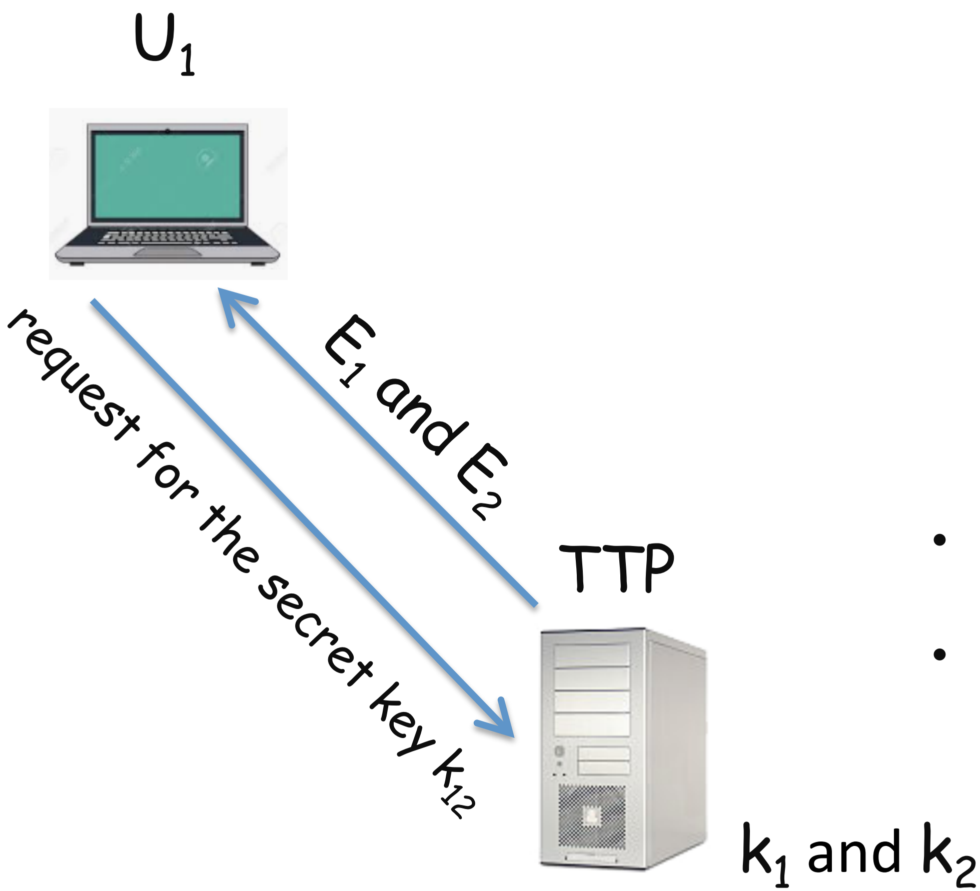
Trusted Third Party

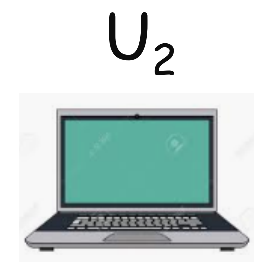


- TTP chooses a random secret k_{12}

Key Exchange

Trusted Third Party

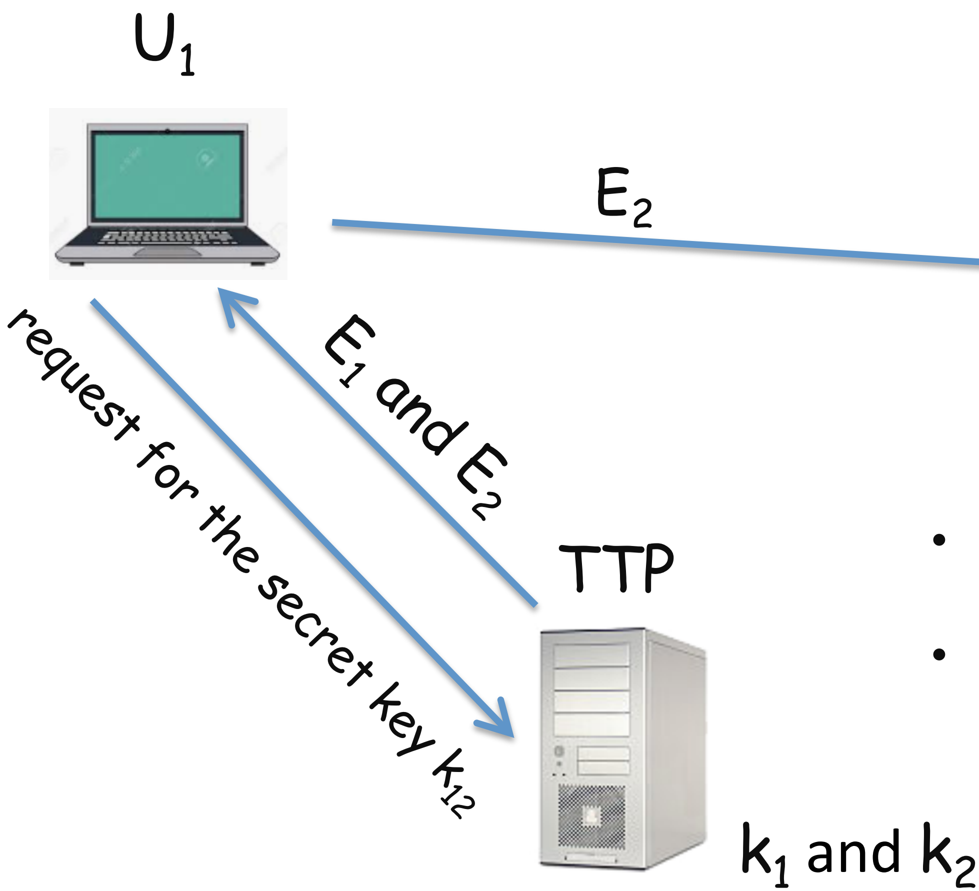




- TTP chooses a random secret k_{12}
- TTP computes
  E_1 = Enc(k_1, 'Users II k_{12}')
  E_2 = Enc(k_2, 'Users II k_{12}')

Key Exchange

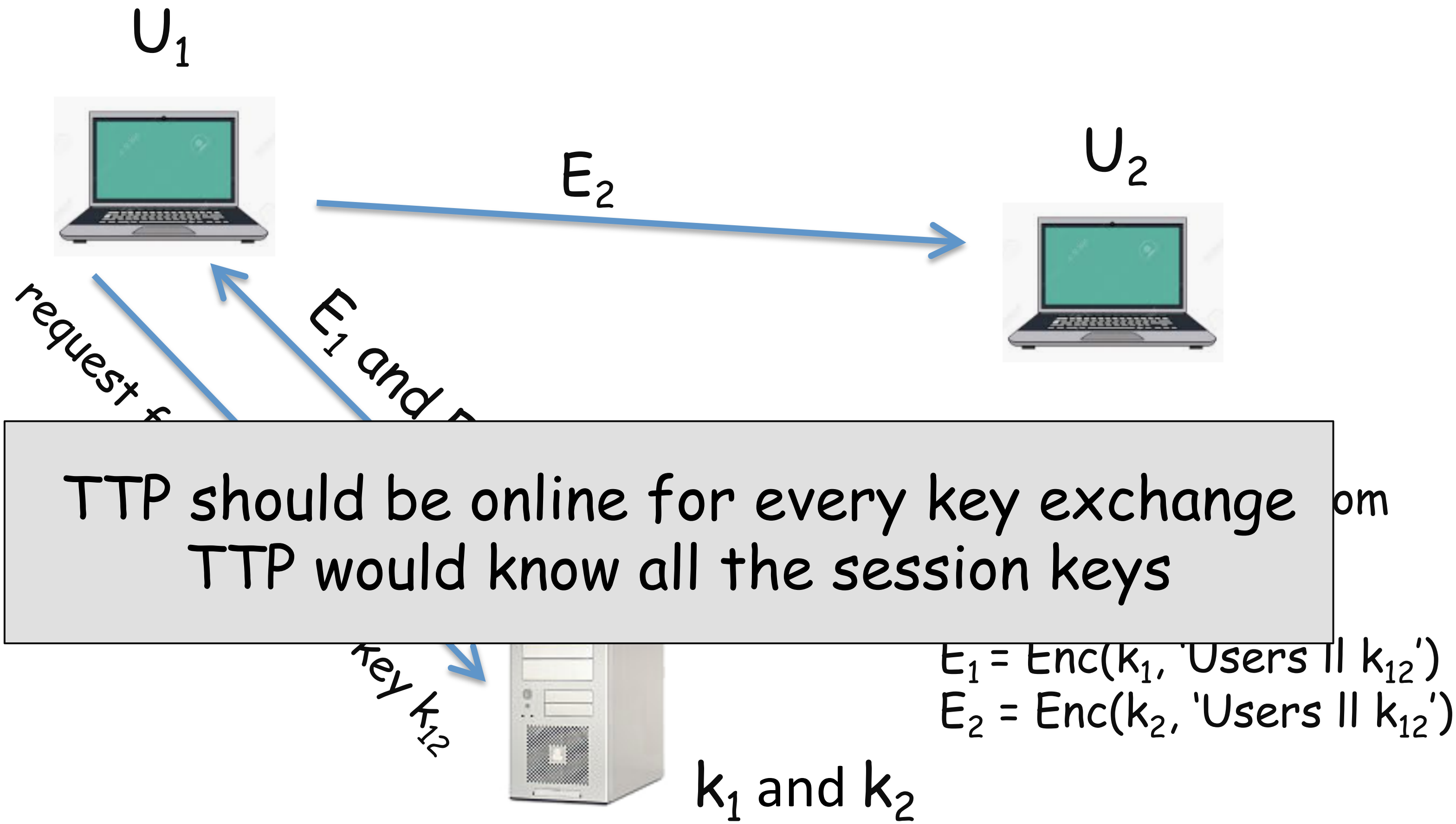
Trusted Third Party



- TTP chooses a random secret k_{12}
- TTP computes
  E_1 = Enc(k_1, 'Users II k_{12}')
  E_2 = Enc(k_2, 'Users II k_{12}')

Key Exchange

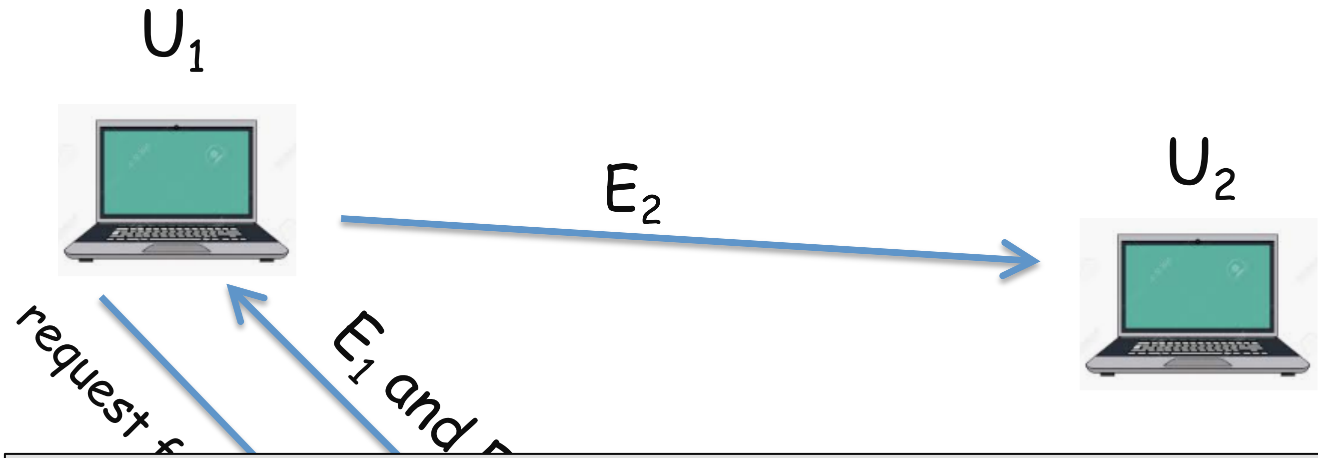
Trusted Third Party



TTP should be online for every key exchange
TTP would know all the session keys

Key Exchange

Trusted Third Party



TTP should be online for every key exchange
TTP would know all the session keys

Can we manage key exchange without an online trusted third party ?

Diffie-Hellman Key Exchange

U_1

U_2

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary

U_1

U_2

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}

U_1

U_2

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}

U_1

U_2

- choose a random a from {1, 2, …, p – 1}

- choose a random b from {1, 2, …, p – 1}

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}

U_1

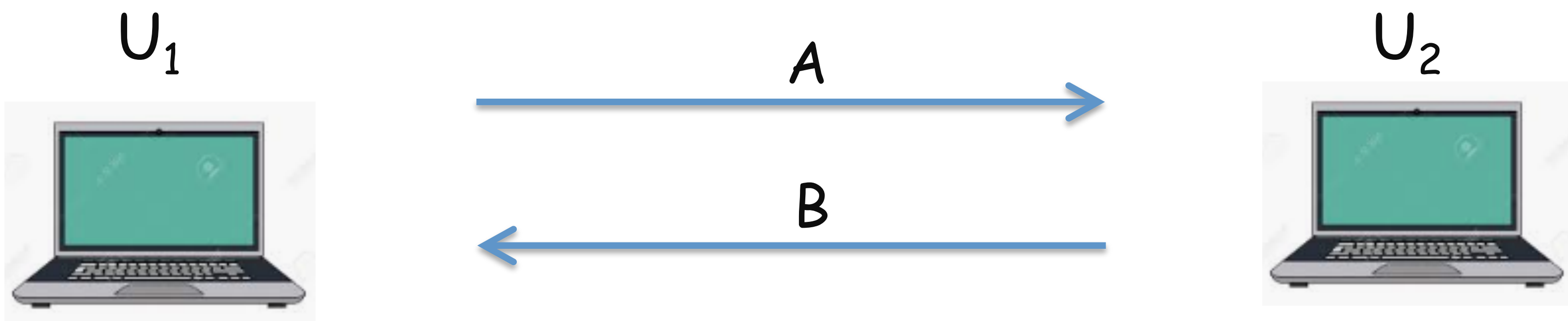
- choose a random a from {1, 2, …, p – 1}
- compute $A=g^a \pmod p$

U_2

- choose a random b from {1, 2, …, p – 1}
- compute $B=g^b \pmod p$

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}

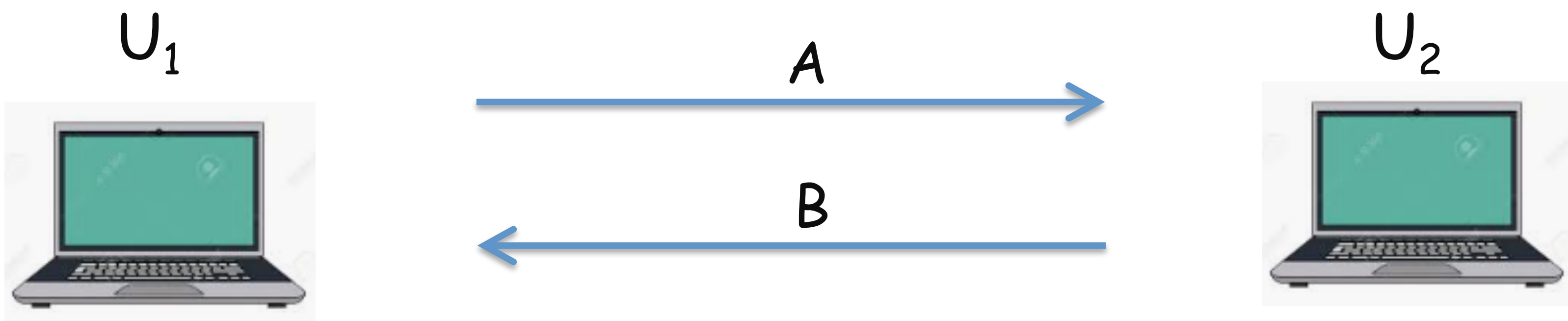


- choose a random a from {1, 2, …, p – 1}
- compute $A=g^a \pmod p$

- choose a random b from {1, 2, …, p – 1}
- compute $B=g^b \pmod p$

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}



- choose a random a from {1, 2, …, p – 1}
- compute $A=g^a \pmod p$
- compute
  $B^a=(g^b)^a=g^{ab} \pmod p$

- choose a random b from {1, 2, …, p – 1}
- compute $B=g^b \pmod p$
- compute
  $A^b=(g^a)^b=g^{ab} \pmod p$

Diffie-Hellman Key Exchange

- U_1 and U_2 want to share a secret through a communication channel eavesdropped by an adversary
- choose a large prime p (2048 bits ≈ 617 digits in current practice)
- choose an integer g from {1, 2, …, p - 1}

U_1

A

B

U_2

$k_{ab} = g^{ab} (\text{mod } p)$

- choose a random a from {1, 2, …, p − 1}
- compute $A = g^a \ (\text{mod } p)$
- compute $B^a = (g^b)^a = g^{ab} (\text{mod } p)$

- …ose a random b …m {1, 2, …, p − 1}
- compute $B = g^b \ (\text{mod } p)$
- compute $A^b = (g^a)^b = g^{ab} (\text{mod } p)$

Security of Diffie-Hellman

Security of Diffie-Hellman

- the adversary gets : p, g, g^a (mod p), g^b (mod p)

Security of Diffie-Hellman

- the adversary gets : p, g, g^a (mod p), g^b (mod p)
- can she compute g^{ab} (mod p) ?

Security of Diffie-Hellman

- the adversary gets : p, g, g^a (mod p), g^b (mod p)
- can she compute g^{ab} (mod p) ?

Diffie-Hellman Function

- $DH_g(g^a, g^b) = g^{ab}$ (mod p)

Security of Diffie-Hellman

- the adversary gets : p, g, g^a (mod p), g^b (mod p)
- can she compute g^{ab} (mod p) ?

Diffie-Hellman Function

- $DH_g(g^a, g^b) = g^{ab}$ (mod p)
- How hard is this function ?

Security of Diffie-Hellman

- the adversary gets : p, g, g^a (mod p), g^b (mod p)
- can she compute g^{ab} (mod p) ?

Diffie-Hellman Function

- $DH_g(g^a, g^b) = g^{ab}$ (mod p)
- How hard is this function ?
  (best known algorithm is General Number Field Sieve that takes $\exp(O(\sqrt[3]{n}))$ –subexponential- for n-bit prime p)

Security of Diffie-Hellman

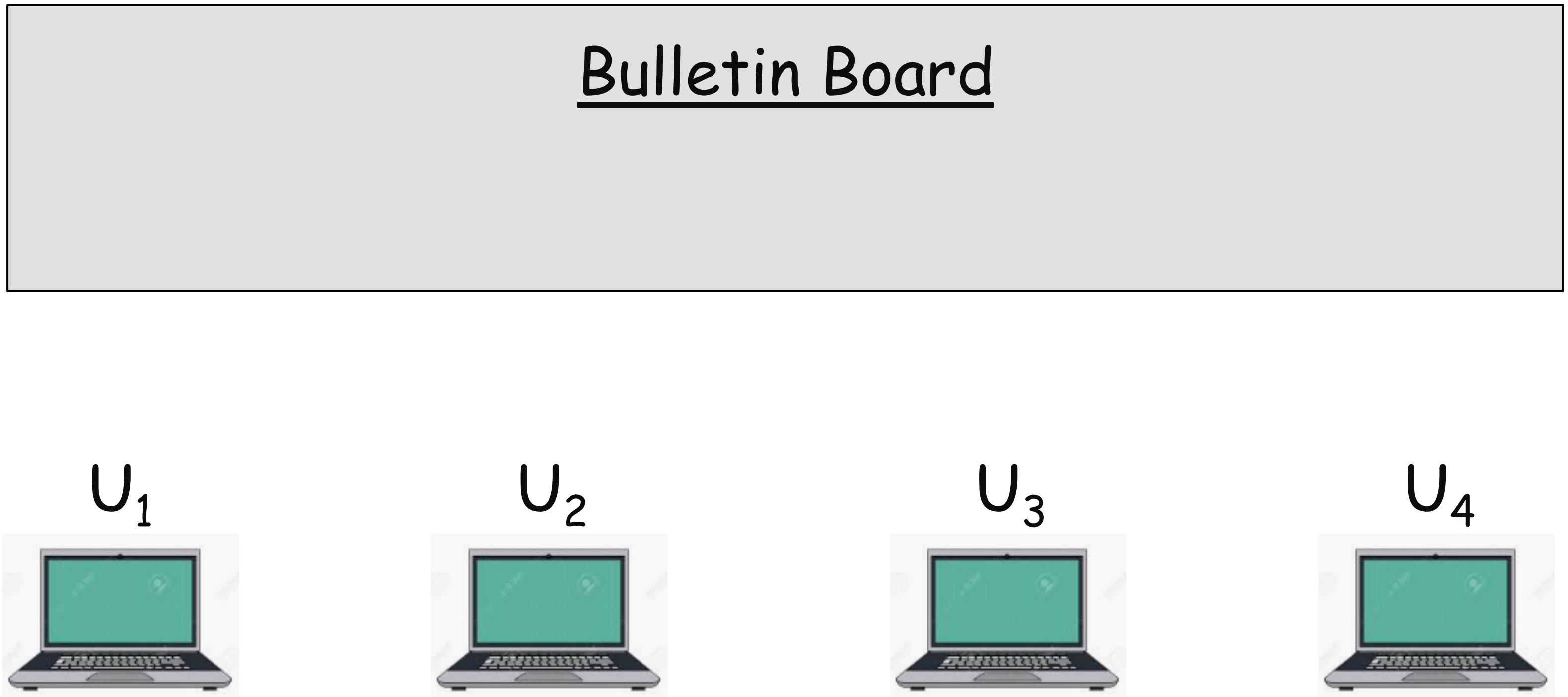
- the adversary gets : p, g, g^a (mod p), g^b (mod p)
- can she compute g^{ab} (mod p) ?

for 1024-bit prime p it is supposed to be e^{10}, however it is ≈ e^{80} (the power has some other constants)

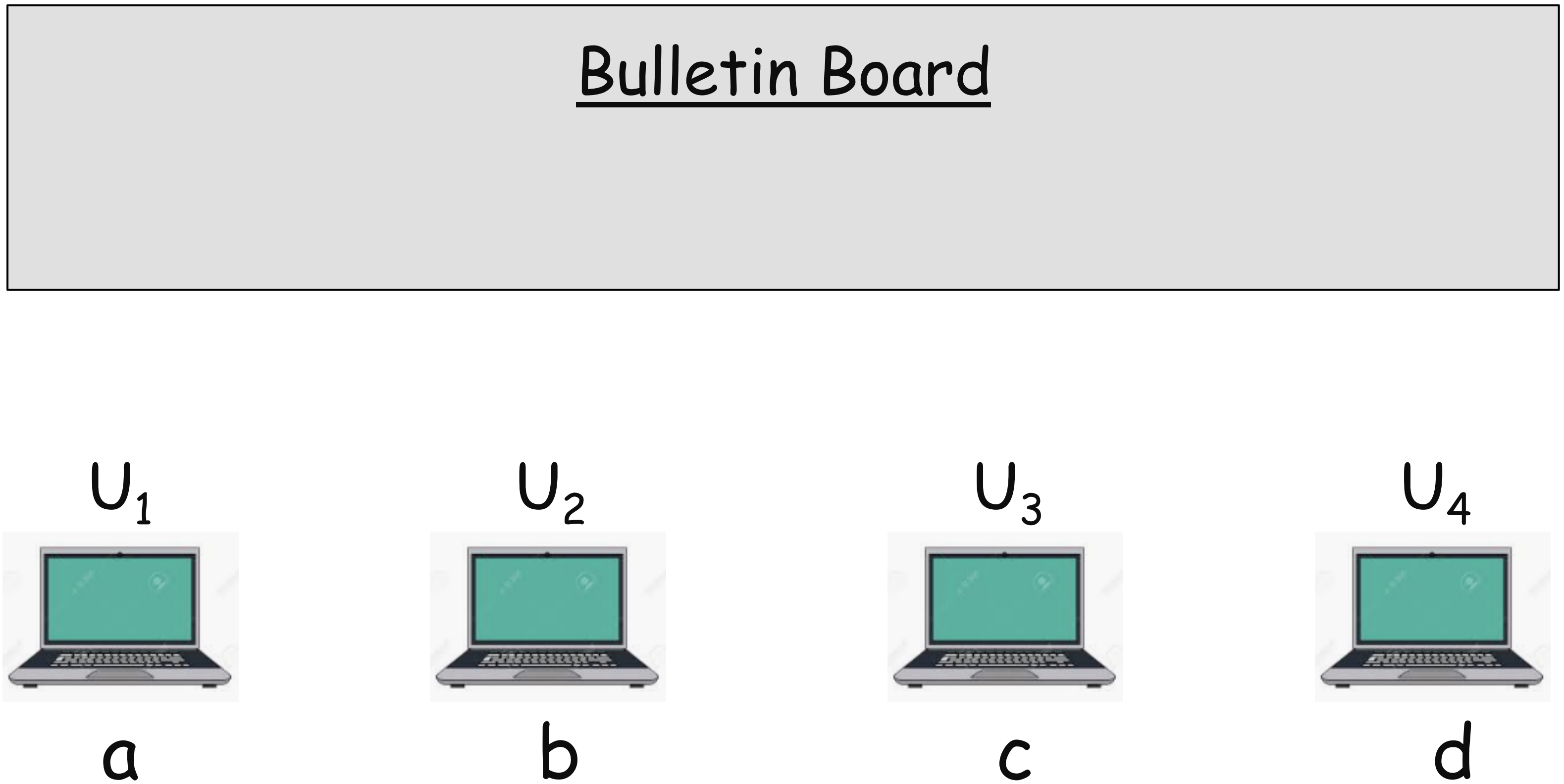
Diffie-Hellman Function

- $DH_g(g^a, g^b) = g^{ab}$ (mod p)
- How hard is this function ?
  (best known algorithm is General Number Field Sieve that takes $\exp(O(\sqrt[3]{n}))$ –subexponential- for n-bit prime p)

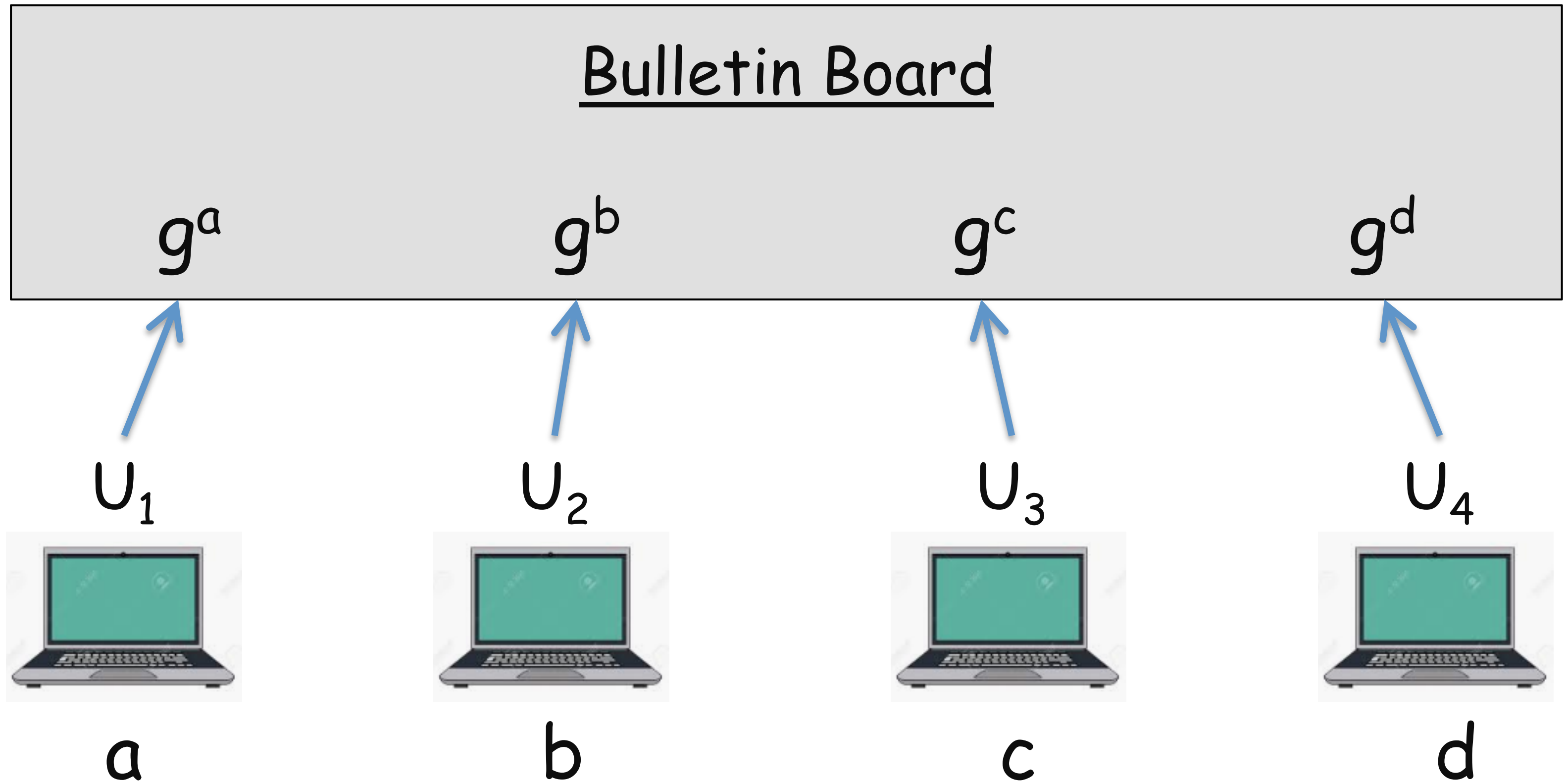
Diffie-Hellman Key Exchange



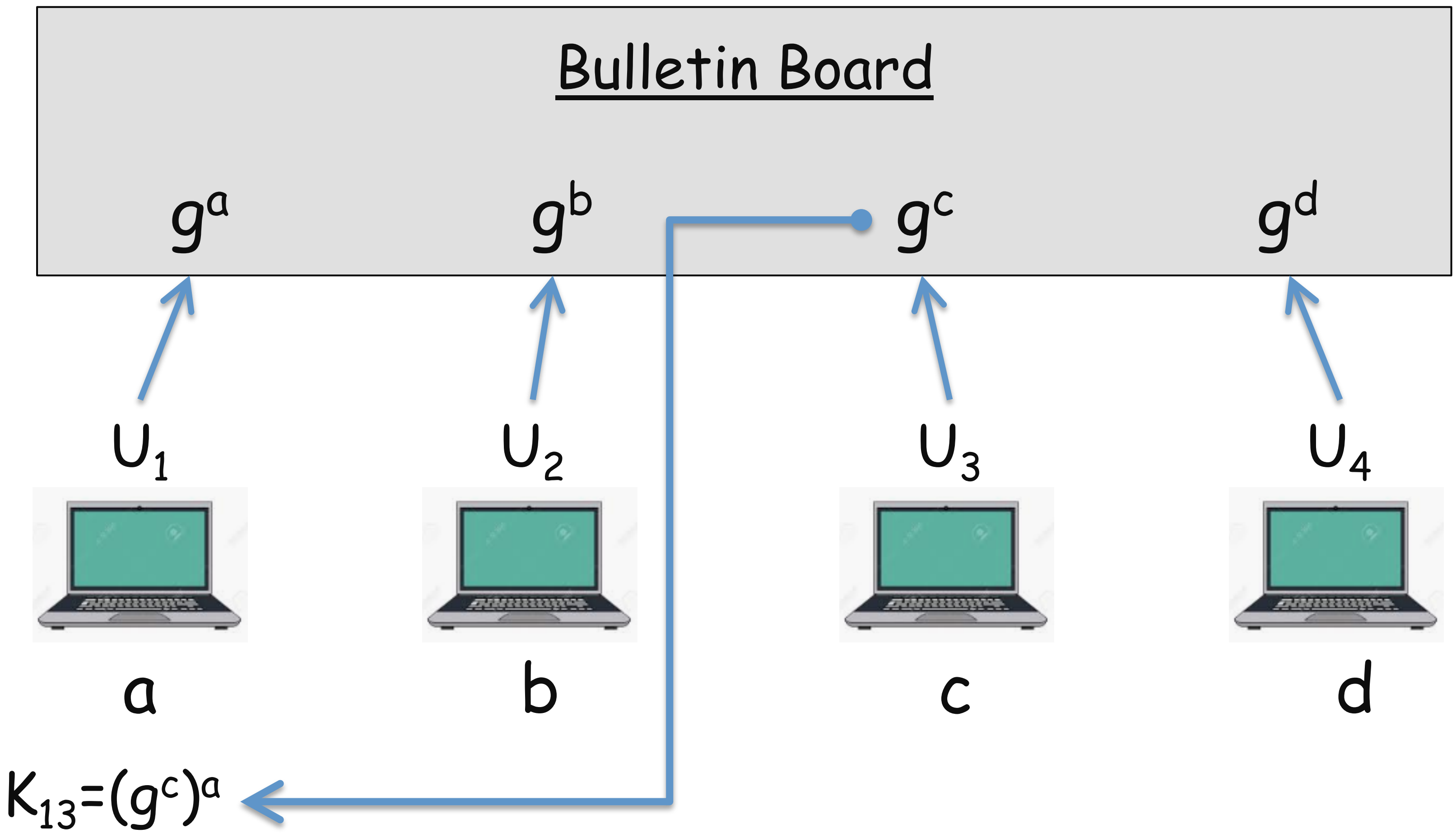
Diffie-Hellman Key Exchange



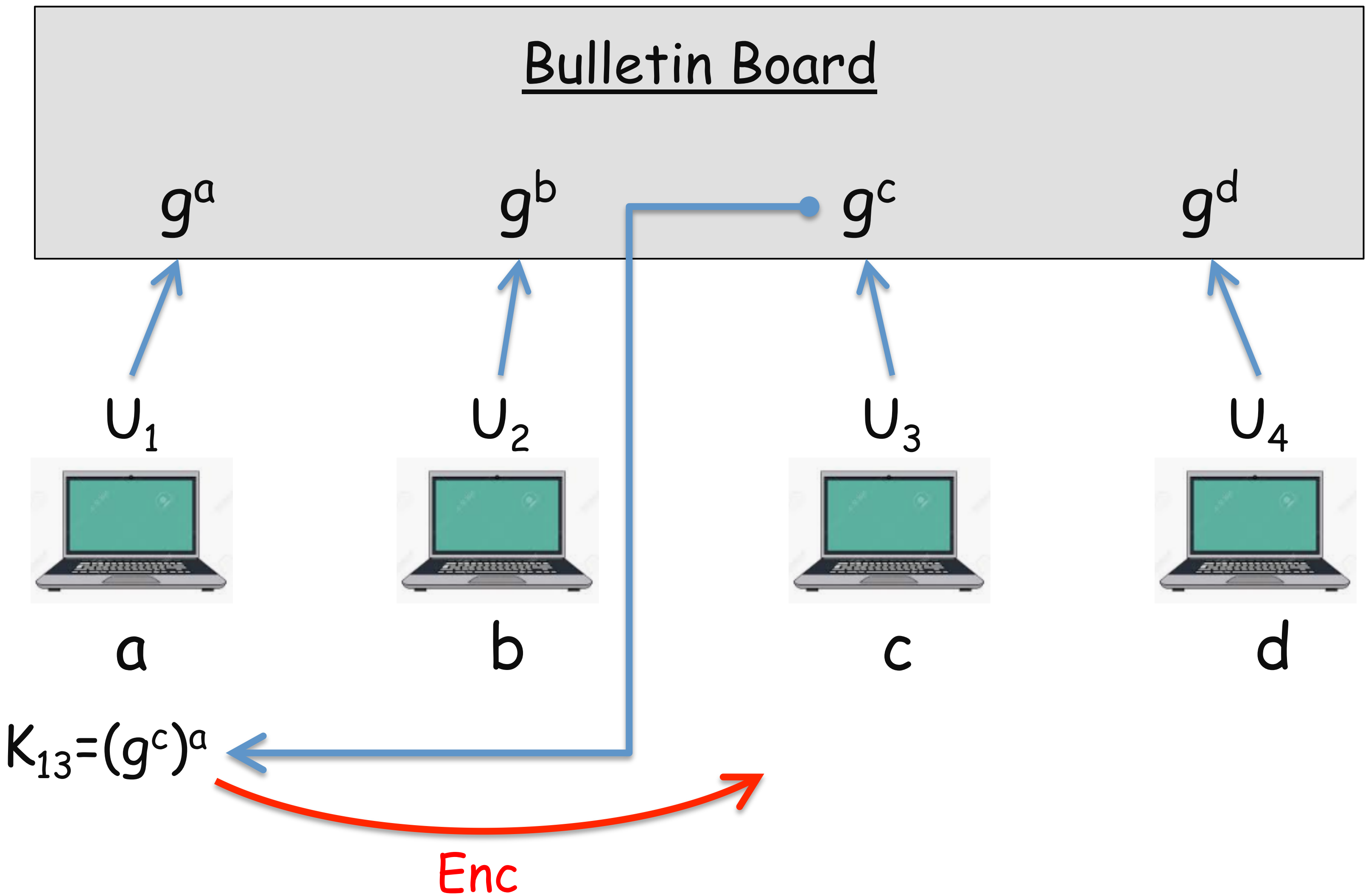
Diffie-Hellman Key Exchange



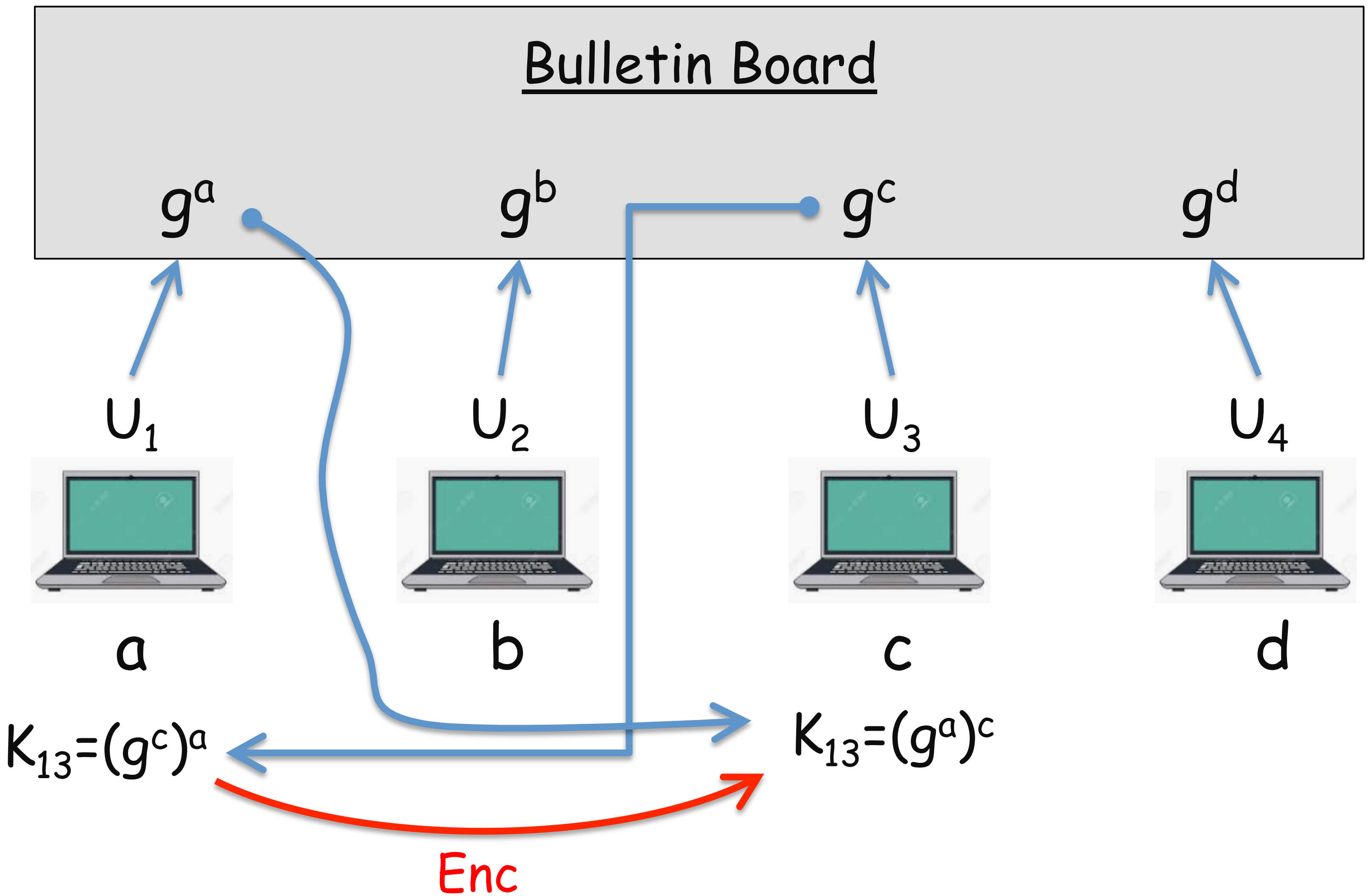
Diffie-Hellman Key Exchange



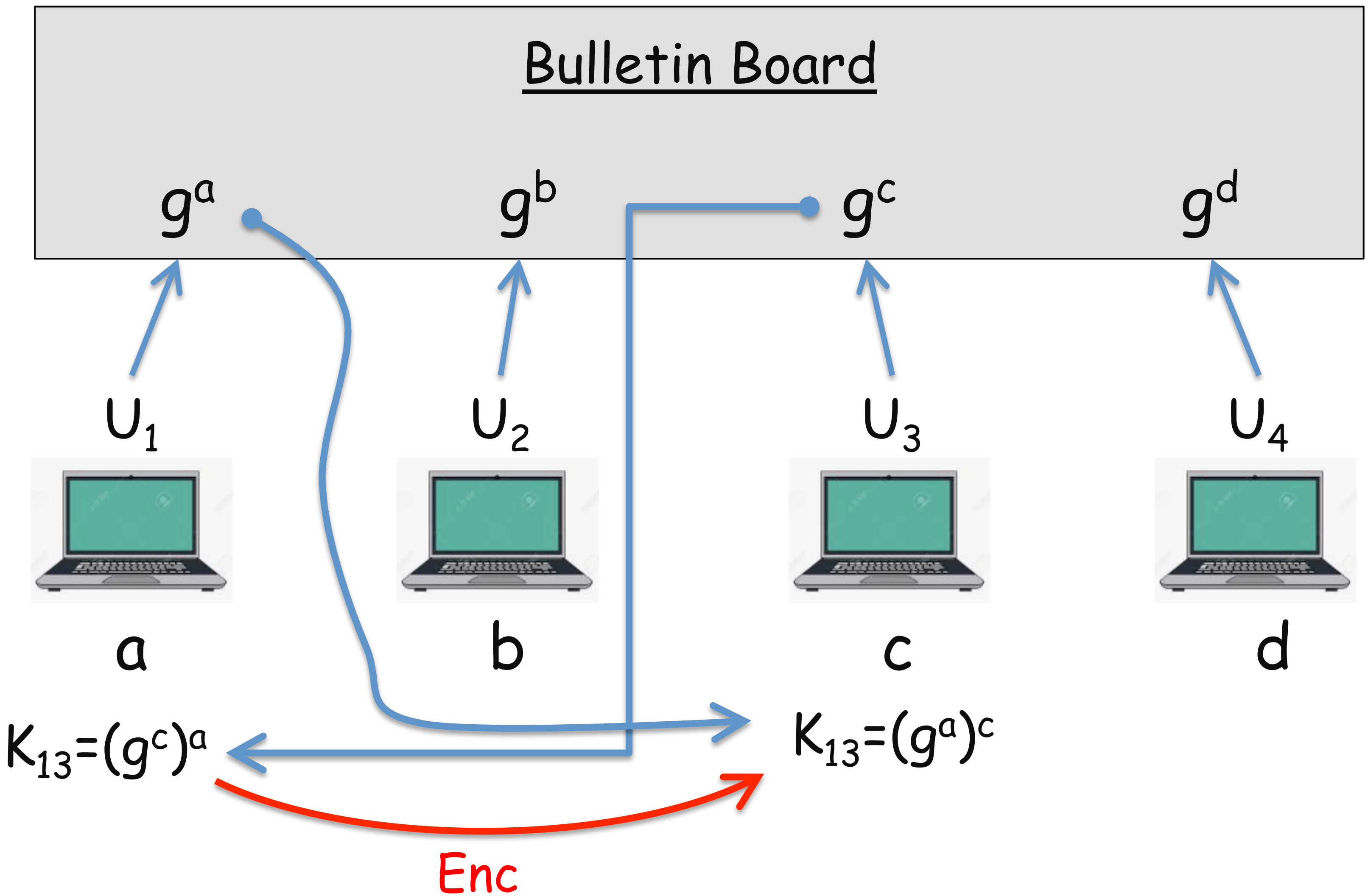
Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

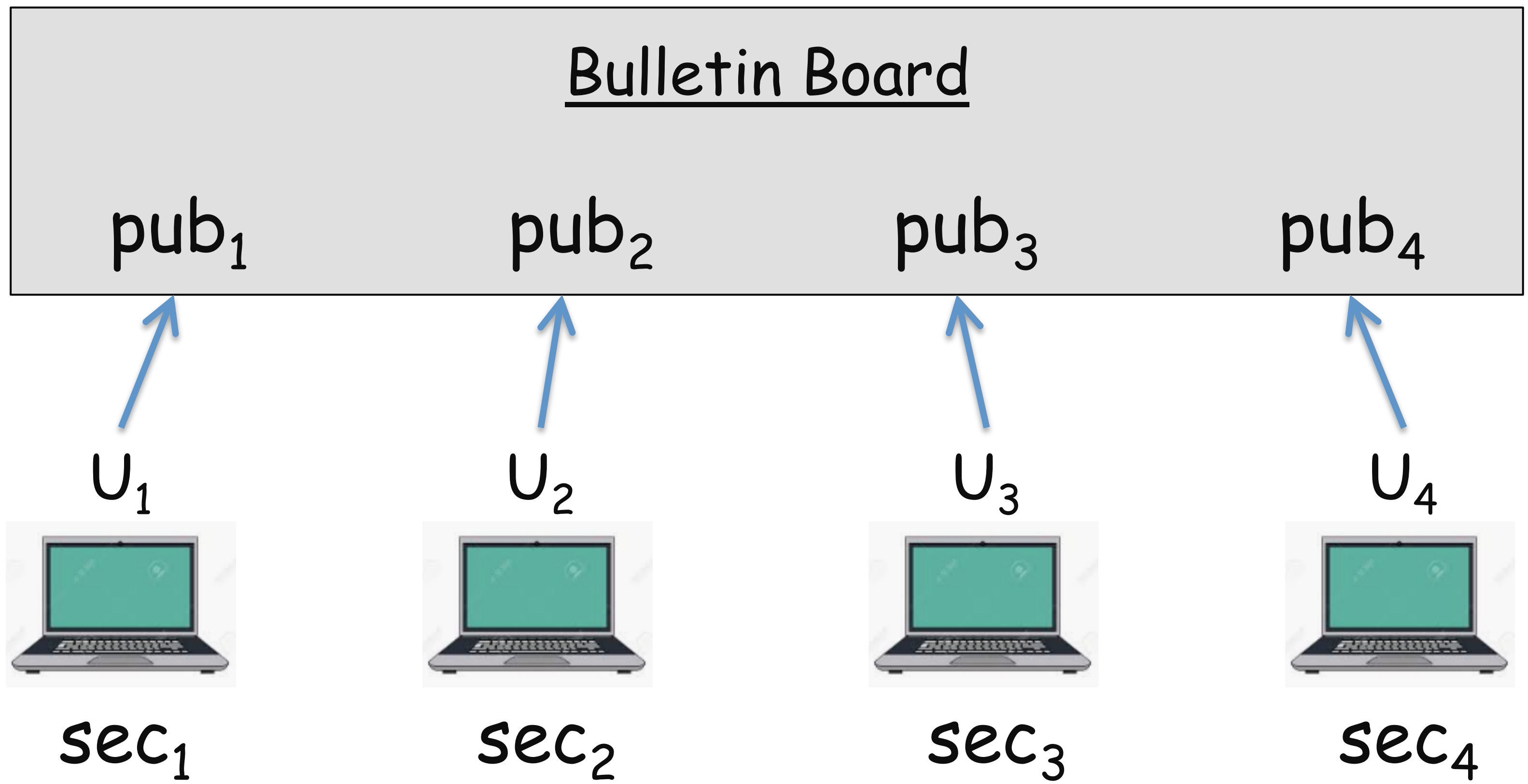


Diffie-Hellman Key Exchange



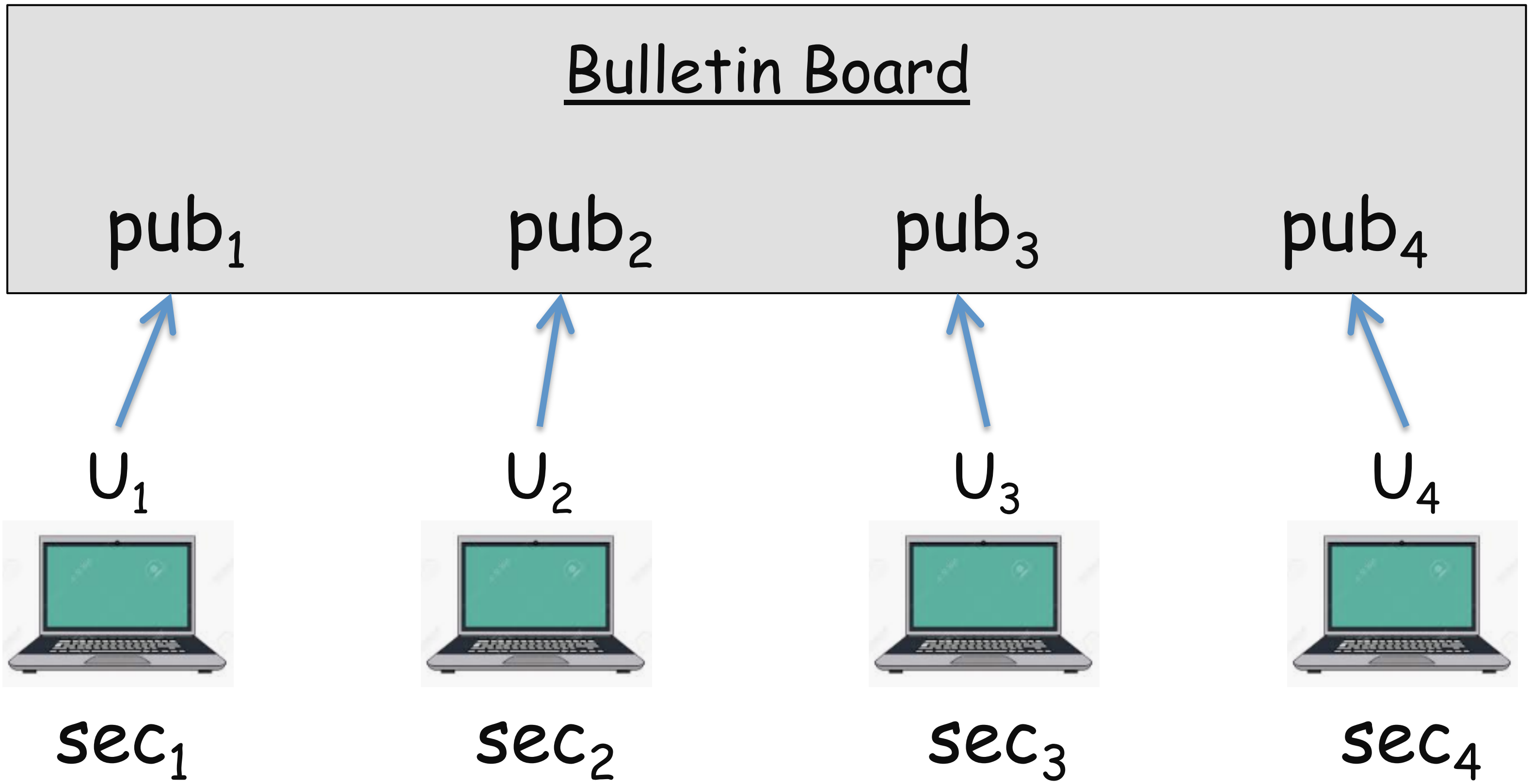
- Users don't need to communicate each other to build the key

Diffie-Hellman Key Exchange



$$K_{1234} \leftarrow (sec_1, pub_2, pub_3, pub_4)$$

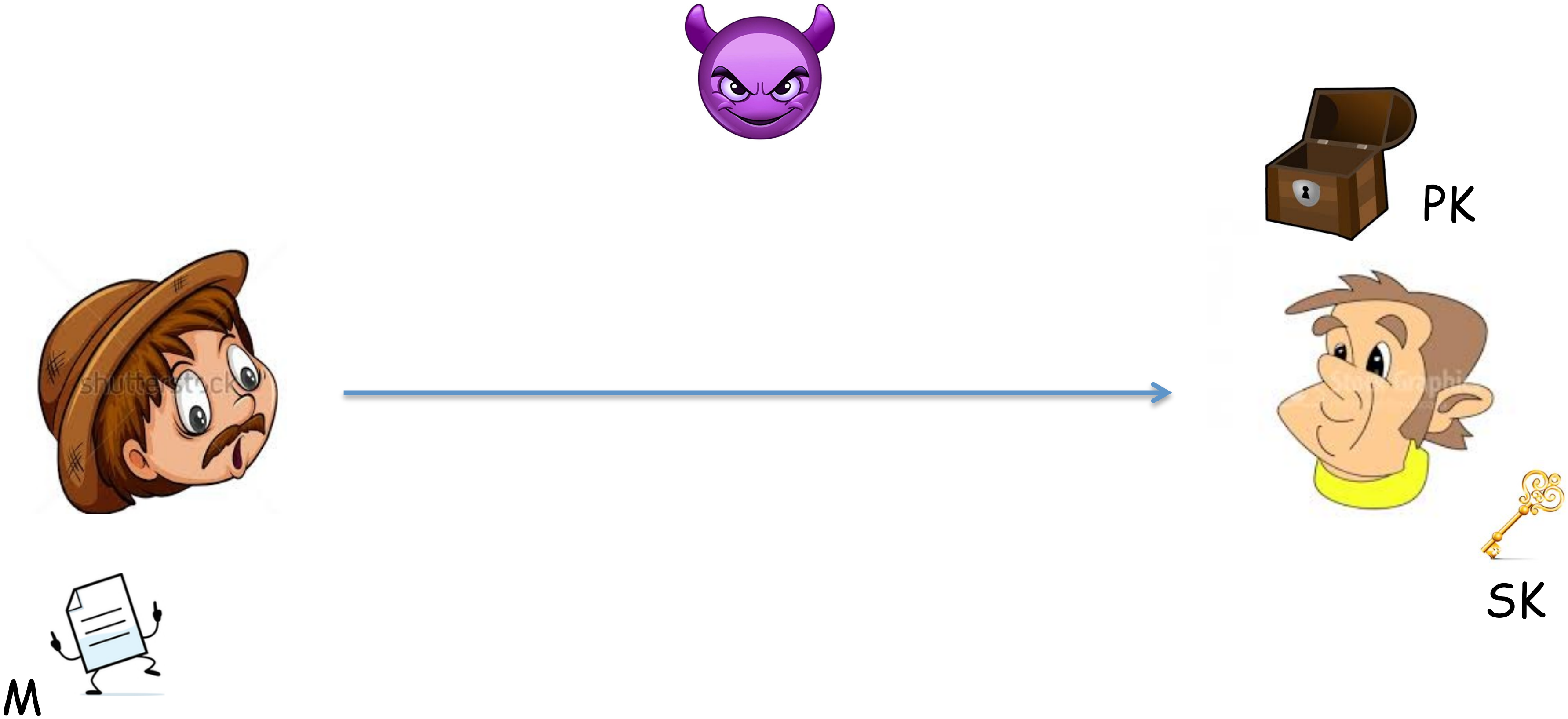
Diffie-Hellman Key Exchange



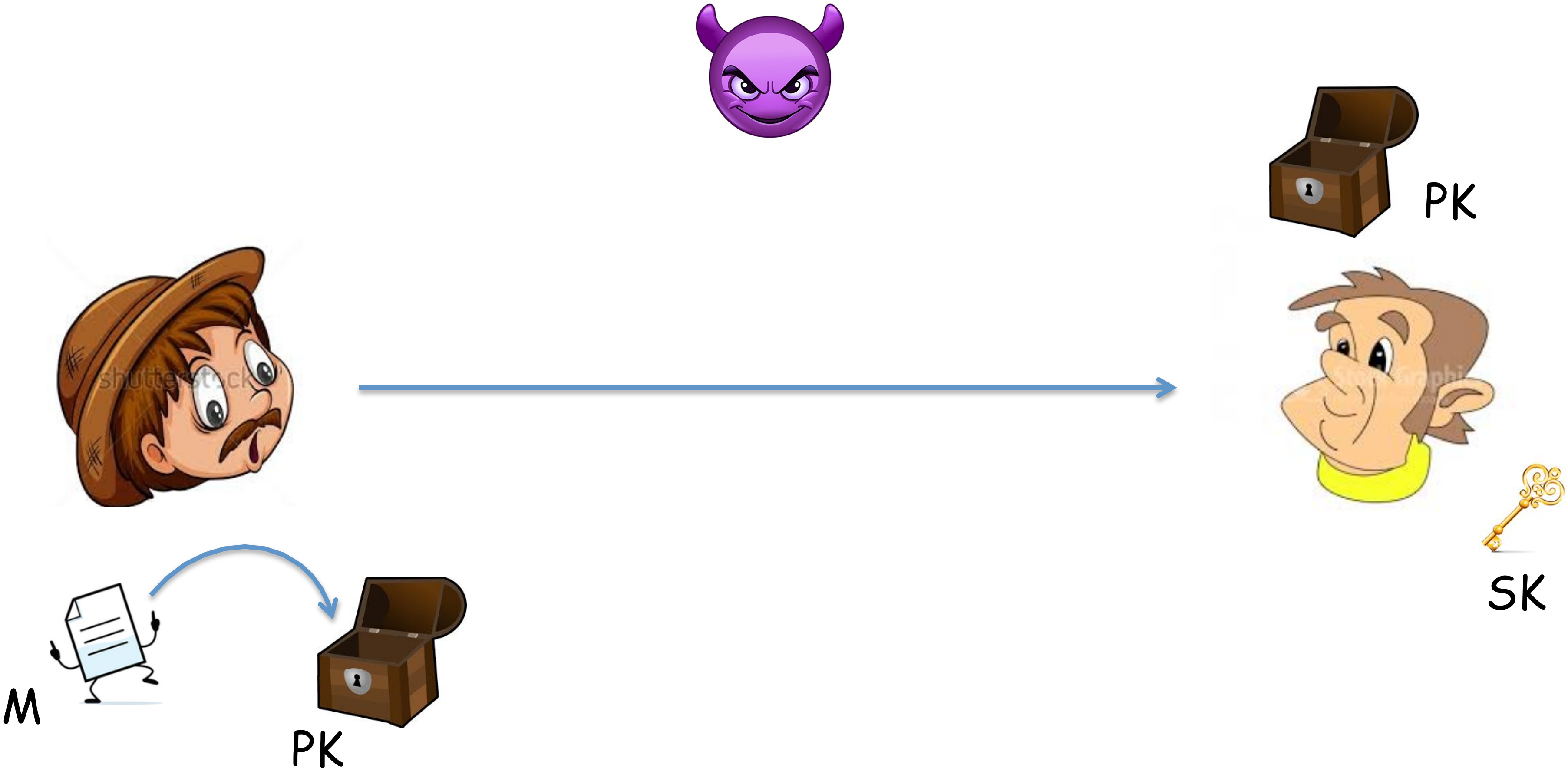
$K_{1234} \leftarrow (sec_1, pub_2, pub_3, pub_4)$

open problem for $n \geq 4$

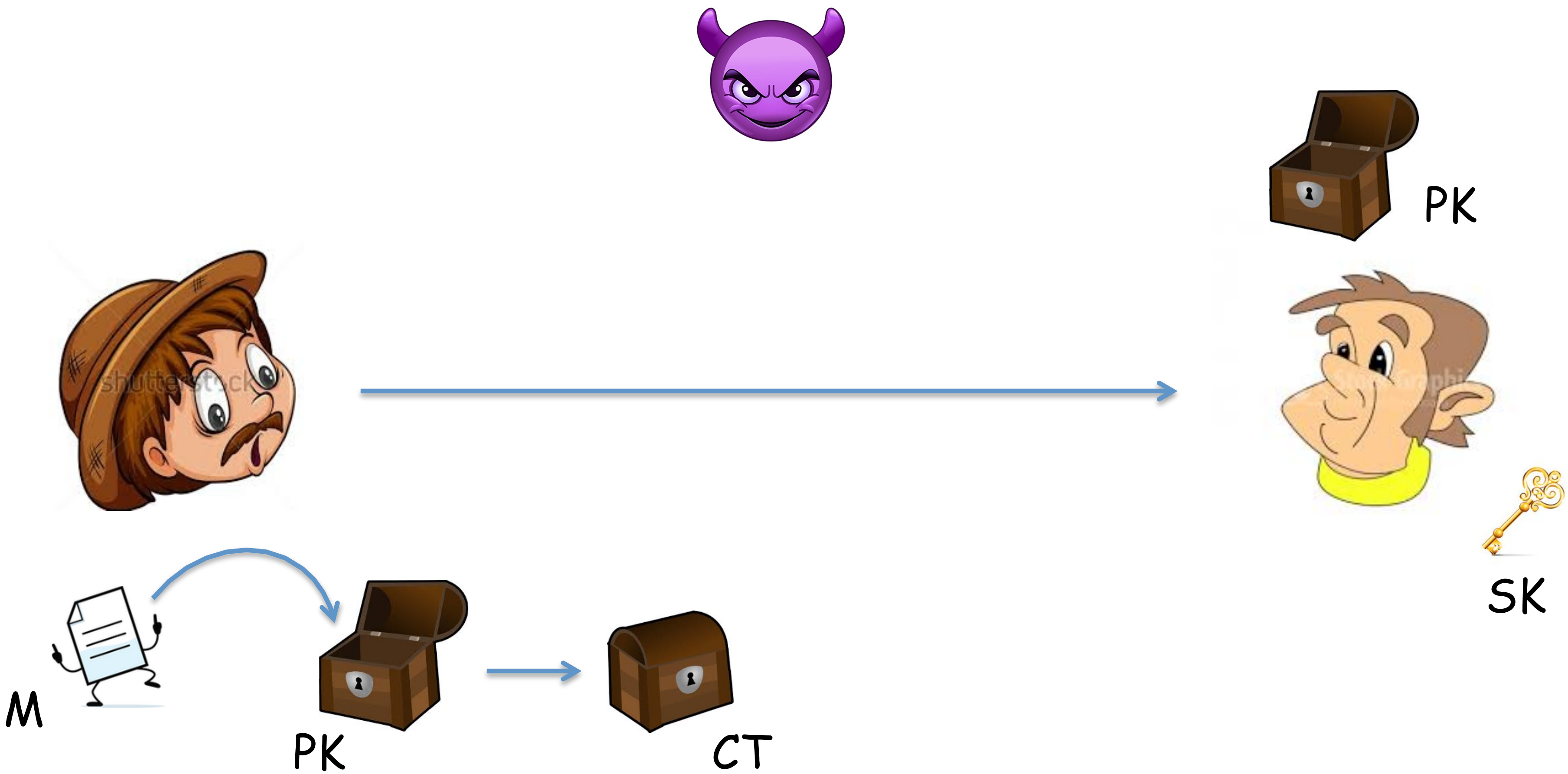
Public-Key Encryption



Public-Key Encryption



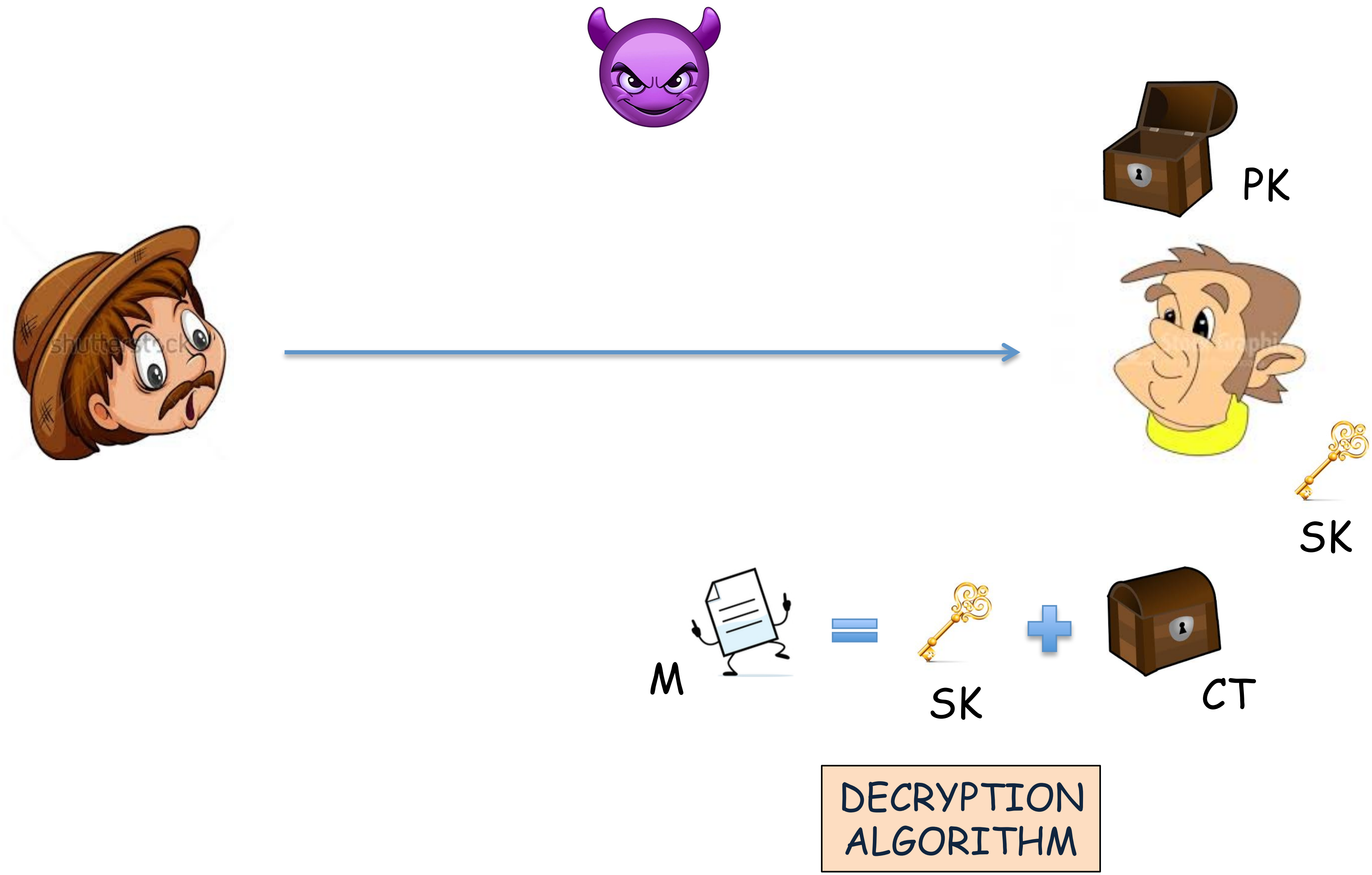
Public-Key Encryption



ENCRYPTION ALGORITHM

Public-Key Encryption

Public-Key Encryption



Public-Key Encryption

A public-key encryption consists of three algorithms

Gen : outputs a key pair (pk, sk)

Enc : takes a message m in M and the public key pk as inputs and outputs a ciphertext c in C

Dec : takes a ciphertext c and the secret key sk as inputs and outputs a message m in M

Correctness

For all (pk, sk) output by Gen and for all m in M

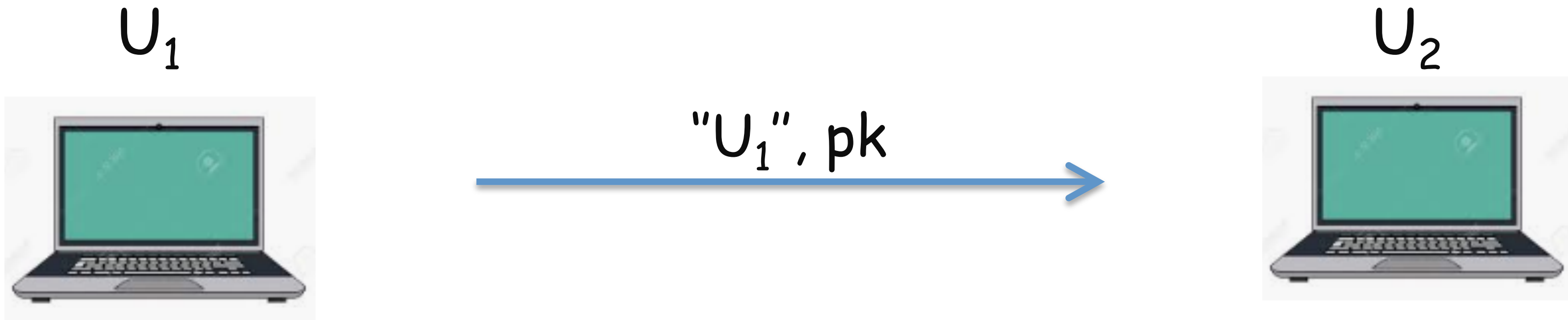
$$\mathrm{Dec}(sk, \mathrm{Enc}(pk, m)) = m$$

Key Exchange using PKE

U_1

U_2

Key Exchange using PKE



$(pk, sk) \leftarrow Gen(.)$

Key Exchange using PKE

U_1

"U_1", pk →

U_2

(pk,sk) ← Gen(.)

- choose a random x
- c ← Enc (pk, x)

Key Exchange using PKE

U_1

"U_1", pk →

← "U_2", c

U_2

(pk,sk) ← Gen(.)

x ← Dec(sk, c)

- choose a random x
- c ← Enc (pk, x)

Key Exchange using PKE

U_1

"U_1", pk →

← "U_2", c

U_2

(pk,sk) ← Gen(.)

x ← Dec(sk, c)

- choose a random x
- c ← Enc (pk, x)

x used as the secret key

Public-Key Encryption

- the idea first introduced by

W. Diffie and M. E. Hellman,
New Directions in Cryptography
IEEE Transaction on Information Theory, 1976

- the first construction introduced by

R. Rivest, A. Shamir, L Adelman
A Method for Obtaining Digital Signatures and Public-Key Cryptosystem
Communications of the ACM, 1978

- security rely on hard problems from number theory and algebra

Factorization Problem, Discrete Logarithm Problem

Public-Key Encryption

- Let N = p.q where p and q are primes

Public-Key Encryption

- Let $N = p.q$ where p and q are primes
- $Z_N = \{0, 1, 2, \ldots, N - 1\}$ and $(Z_N)^*$: the set of all invertible elements in Z_N

Public-Key Encryption

- Let $N = p.q$ where p and q are primes
- $Z_N = \{0, 1, 2, \ldots, N - 1\}$ and $(Z_N)^*$: the set of all invertible elements in Z_N
- if x in $(Z_N)^*$, then $gcd(x, N) = 1$

Public-Key Encryption

- Let $N = p.q$ where p and q are primes
- $Z_N = \{0, 1, 2, \ldots, N - 1\}$ and $(Z_N)^*$: the set of all invertible elements in Z_N
- if x in $(Z_N)^*$, then $gcd(x, N) = 1$
- $|(Z_N)^*| = phi(N) = (p - 1)(q - 1)$

Public-Key Encryption

- Let $N = p.q$ where p and q are primes
- $Z_N = \{0, 1, 2, ..., N - 1\}$ and $(Z_N)^*$: the set of all invertible elements in Z_N
- if x in $(Z_N)^*$, then $gcd(x, N) = 1$
- $| (Z_N)^* | = phi(N) = (p - 1)(q - 1)$

> - $Z_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
> - $(Z_{10})^* = \{1, 3, 7, 9\}$, $3.7 = 1 \pmod{10}$
> - $| (Z_{10})^* | = phi(N) = (2 - 1)(5 - 1) = 4$

Public-Key Encryption

- Let $N = p.q$ where p and q are primes
- $Z_N = \{0, 1, 2, \ldots, N - 1\}$ and $(Z_N)^*$: the set of all invertible elements in Z_N
- if x in $(Z_N)^*$, then $\gcd(x, N) = 1$
- $|(Z_N)^*| = \text{phi}(N) = (p - 1)(q - 1)$
- Euler Theorem

$$\text{for all } x \text{ in } (Z_N)^*,\ x^{\text{phi}(N)} = 1 \pmod N$$

RSA

KeyGen

- pick two large primes p and q
- compute N = p.q

RSA

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1

RSA

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)

RSA

for the equation a.x = 1 mod N

if gcd(a,N) = 1, then there is a unique solution

<u>KeyGen</u>

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)

RSA

PK=(N,e)

SK=(d,p,q)

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)
- keep (d, p, q) as secret key, and publish (N, e) as public key

RSA

PK=(N,e)

SK=(d,p,q)

RSA

PK=(N,e)

SK=(d,p,q)

Encryption

$c = m^e \pmod{N}$ where m in $(Z_N)^*$

RSA

PK=(N,e)

c

SK=(d,p,q)

Encryption

$c = m^e \pmod{N}$ where m in $(Z_N)^*$

RSA

PK=(N,e)

c

SK=(d,p,q)

Decryption

$c^d \pmod{N}$

RSA

PK=(N,e)

c

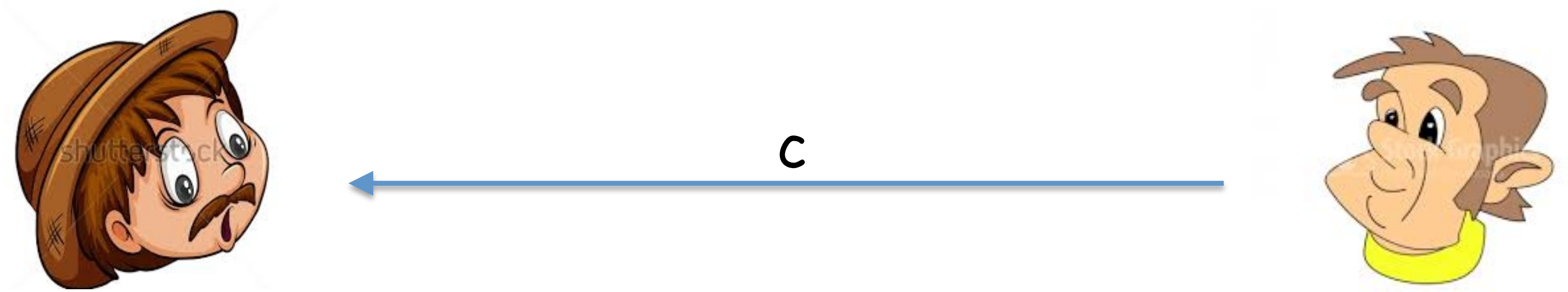
SK=(d,p,q)

Decryption

$c^d \pmod N = m^{ed} \pmod N$

RSA

PK=(N,e)



SK=(d,p,q)

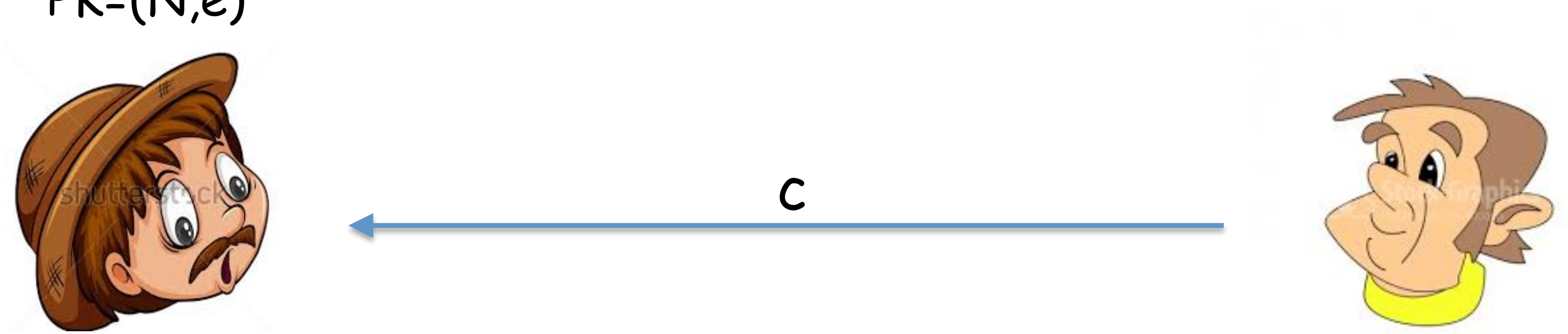
Decryption

$e.d = 1 \bmod phi(N)$

$$c^d \pmod N = m^{ed} \pmod N$$
$$= m^{1 + k.phi(N)} \pmod N$$

RSA

PK=(N,e)



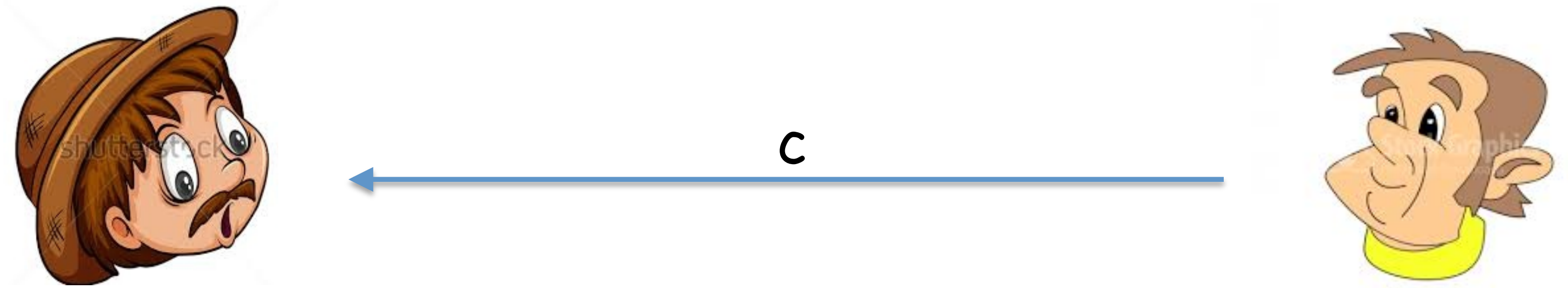
SK=(d,p,q)

Decryption

$$
\begin{aligned}
c^d \pmod N &= m^{ed} \pmod N \\
&= m^{1 + k.phi(N)} \pmod N \\
&= m \,.\, m^{phi(N).k} \pmod N
\end{aligned}
$$

RSA

PK=(N,e)



SK=(d,p,q)

Decryption

$$x^{phi(N)} = 1 \pmod N$$

$$\begin{aligned} c^d \pmod N &= m^{ed} \pmod N \\ &= m^{1 + k.phi(N)} \pmod N \\ &= m \,.\, m^{phi(N).k} \pmod N \\ &= m \pmod N \end{aligned}$$

RSA in Practice

- If you factor N, you can break RSA

RSA in Practice

- If you factor N, you can break RSA

  if you have N = p.q, then you can compute

$$\Phi(N)=(p-1)(q-1)$$

RSA in Practice

- If you factor N, you can break RSA

  if you have $N = p.q$, then you can compute

$$\Phi(N)=(p-1)(q-1)$$

  if you have $\Phi(N)$, then you can find the secret key d by

  solving the equation $e.d = 1 \pmod{\Phi(N)}$

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

$$Y = 13.24 \times D^{1/3} + 1928.6$$

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

$$Y = 13.24 \times D^{1/3} + 1928.6$$

- According to the formula :

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

$$Y = 13.24 \times D^{1/3} + 1928.6$$

- According to the formula :

512-bit number would be factored by 1999
(RSA-155 [512-bit] was factored by Lenstra in 1999

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

$$Y = 13.24 \times D^{1/3} + 1928.6$$

- According to the formula :

512-bit number would be factored by 1999
(RSA-155 [512-bit] was factored by Lenstra in 1999
768-bit number would be factored by 2010
(RSA-768 [232 digits] was factored by Lenstra in 2009

RSA in Practice

Factorization

- Let D be the number of decimal digits, Y be the year the factorization occurs. From the running time of NFS and assuming Moore's law, Brent derived a formula

$$Y = 13.24 \times D^{1/3} + 1928.6$$

- According to the formula :

512-bit number would be factored by 1999
(RSA-155 [512-bit] was factored by Lenstra in 1999
768-bit number would be factored by 2010
(RSA-768 [232 digits] was factored by Lenstra in 2009
1024-bit number would be factored by 2018
2048-bit number would be factored by 2041

RSA in Practice

RSA-768 [232 digits] was factored by Lenstra in 2009

- They spent half a year on 80 processors on polynomial selection. This was about 3% of the main task, the sieving, which was done on many hundreds of machines and took almost two years.
- On a single core 2.2 GHz AMD Opteron processor with 2 GB RAM, sieving would have taken about fifteen hundred years.

RSA in Practice

RSA-768 [232 digits] was factored by Lenstra in 2009

- They spent half a year on 80 processors on polynomial selection. This was about 3% of the main task, the sieving, which was done on many hundreds of machines and took almost two years.
- On a single core 2.2 GHz AMD Opteron processor with 2 GB RAM, sieving would have taken about fifteen hundred years.
- Factoring a 1024-bit RSA modulus would be about a thousand times harder, and a 768-bit RSA modulus is several thousands times harder to factor than a 512-bit one
- They suggest to leave 1024-bit modulus within the next three to four years (by 2013-2014)

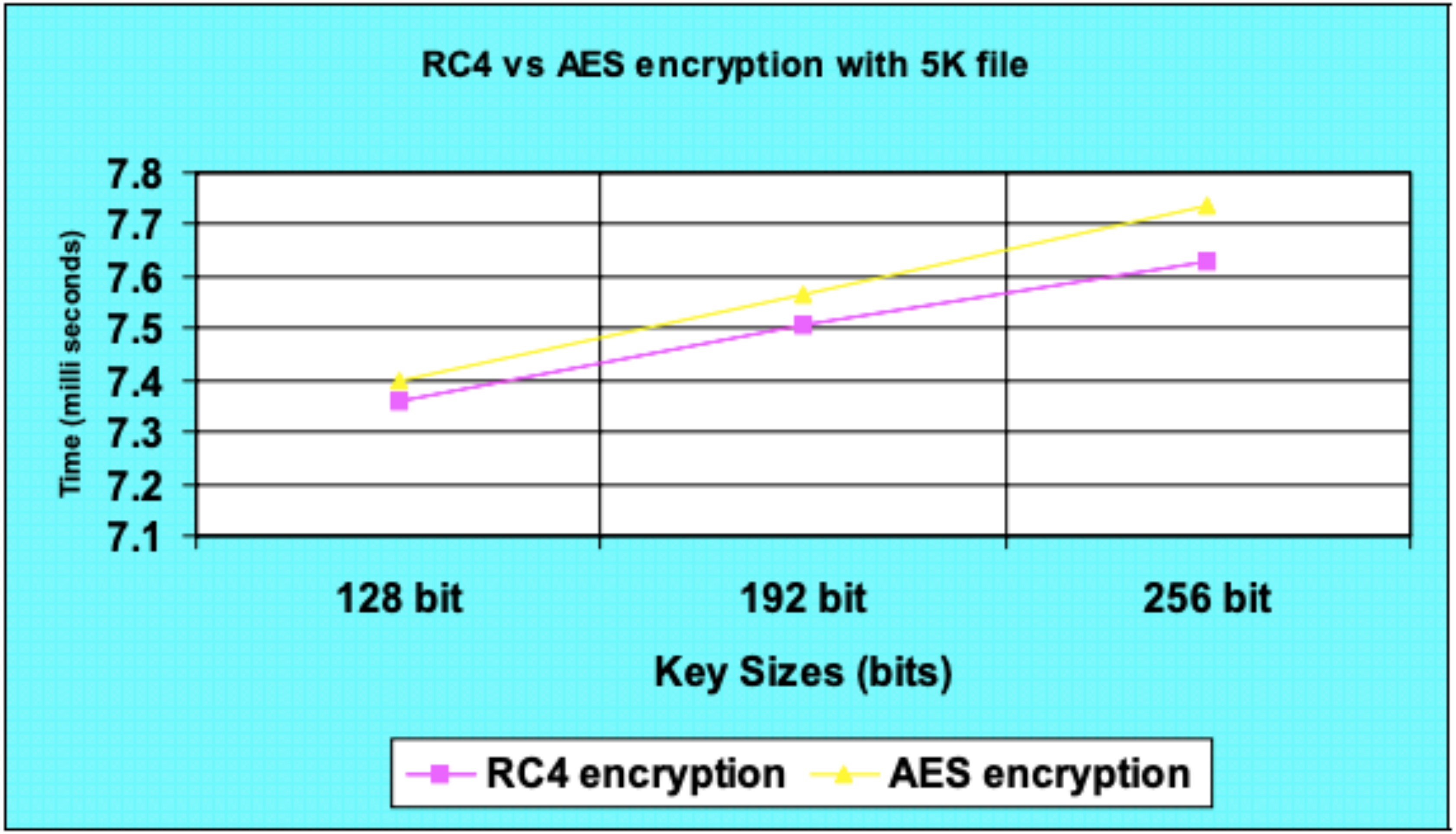
RSA in Practice

Cryptographic Algorithm	Type	Purpose	Impact from large-scale quantum computer
AES	Symmetric key	Encryption	Larger key sizes needed
SHA-2, SHA-3	---------------	Hash functions	Larger output needed
RSA	Public key	Signatures, key establishment	No longer secure
ECDSA, ECDH (Elliptic Curve Cryptography)	Public key	Signatures, key exchange	No longer secure
DSA (Finite Field Cryptography)	Public key	Signatures, key exchange	No longer secure

AES key size : 80 bits
128 bits
256 bits

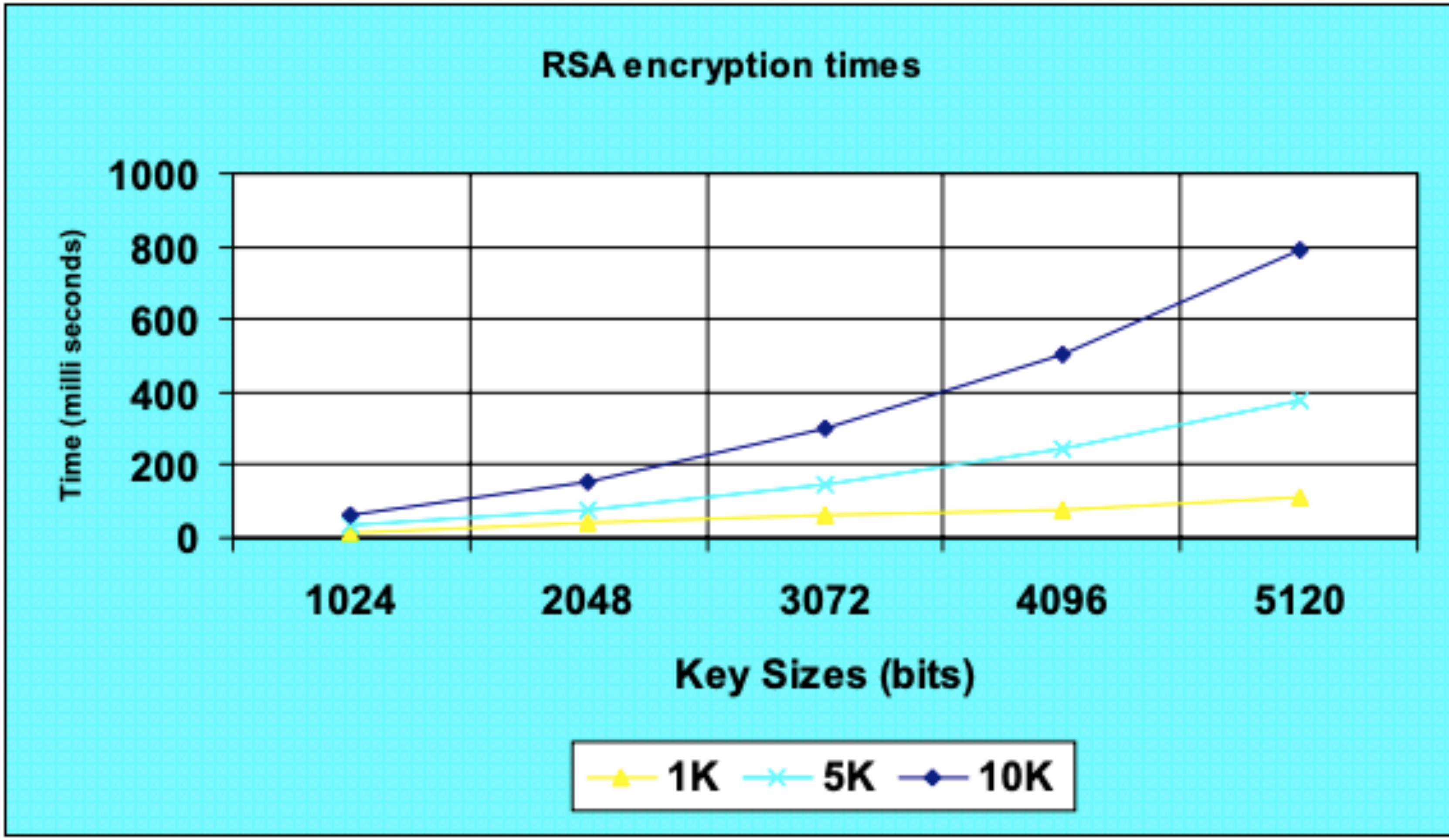
RSA modulus size : 1024 bits
3072 bits
15360 bits

RSA in Practice



- AES-128 for 5K file : 7.40 ms
- AES-192 for 5K file : 7.55 ms
- AES-256 for 5K file : 7.73 ms

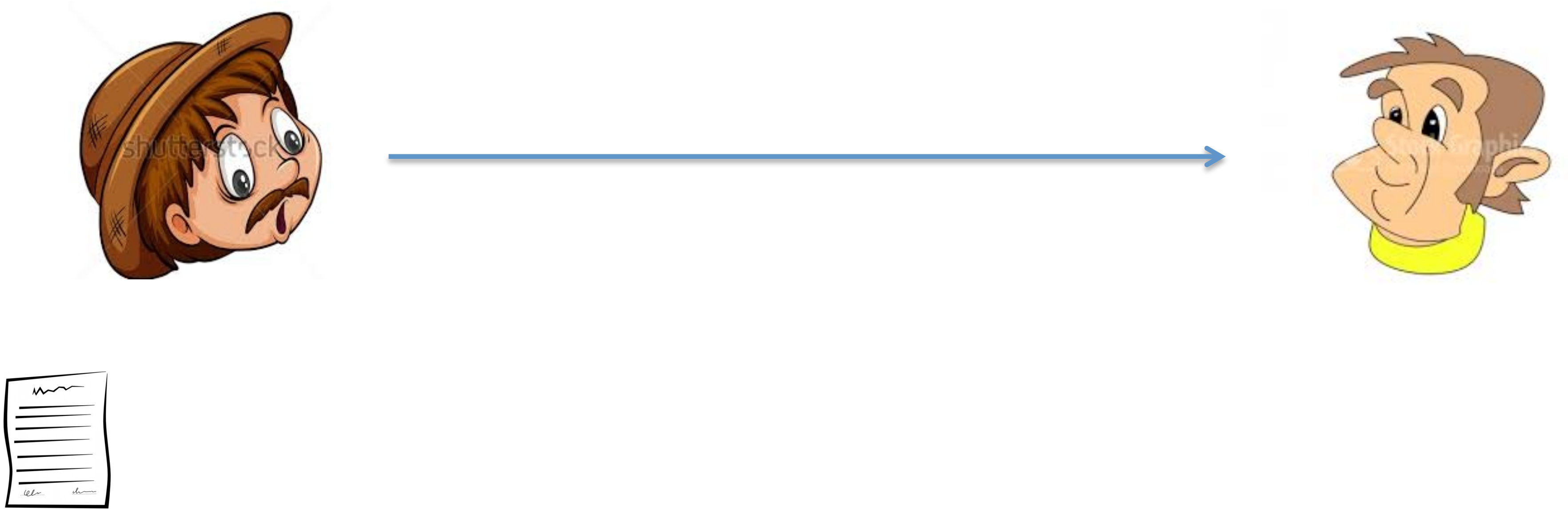
RSA in Practice



- RSA-1024 for 5K file : 50 ms
- RSA-2048 for 5K file : 100 ms
- RSA-3072 for 5K file : 150 ms

Digital Signature Scheme

signing by hand

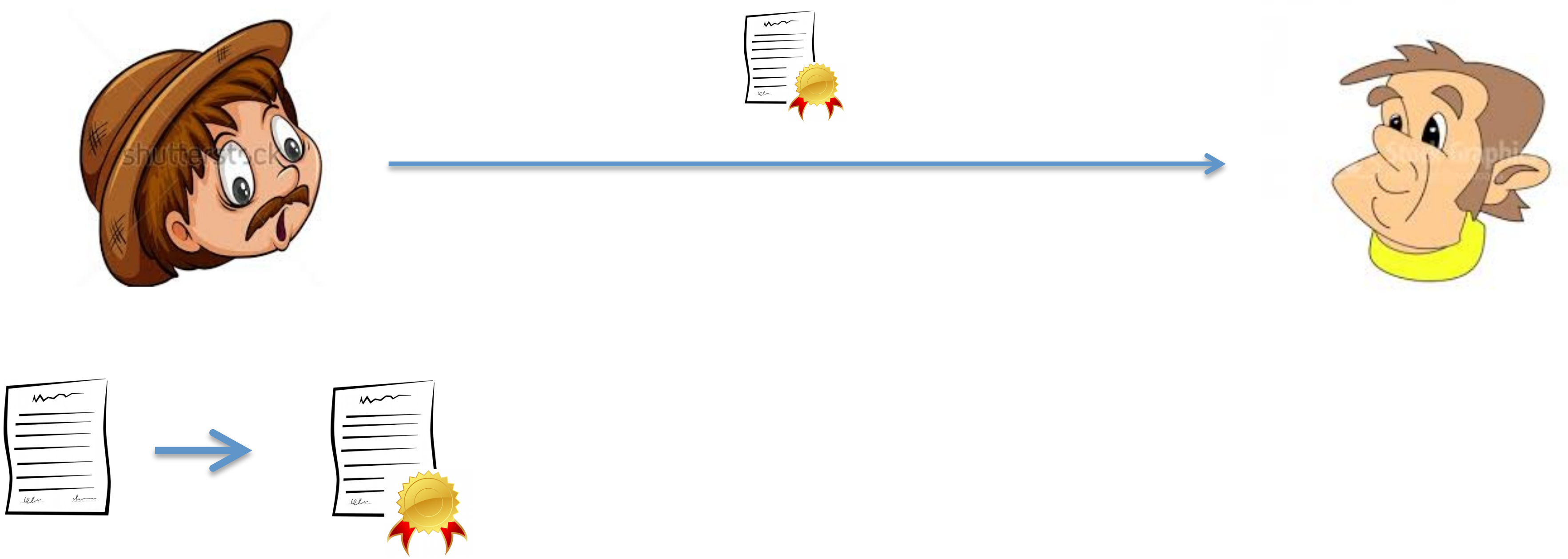


Digital Signature Scheme

signing by hand

Digital Signature Scheme

signing by hand

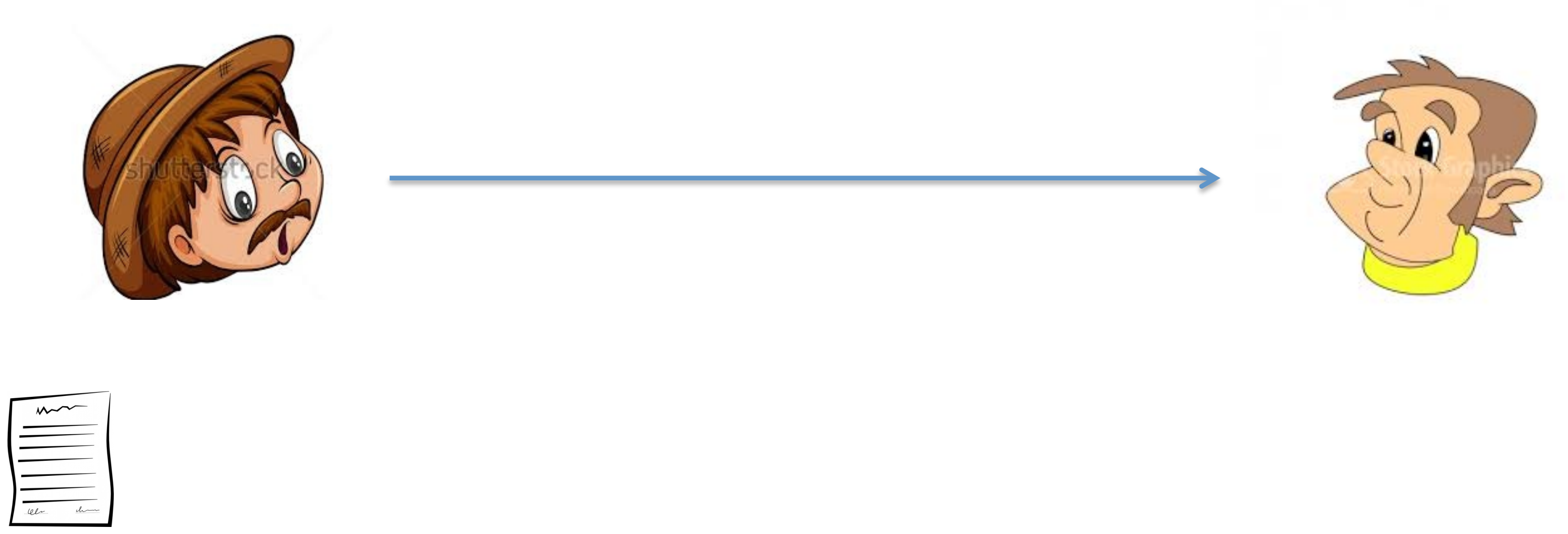


Digital Signature Scheme

signing by hand

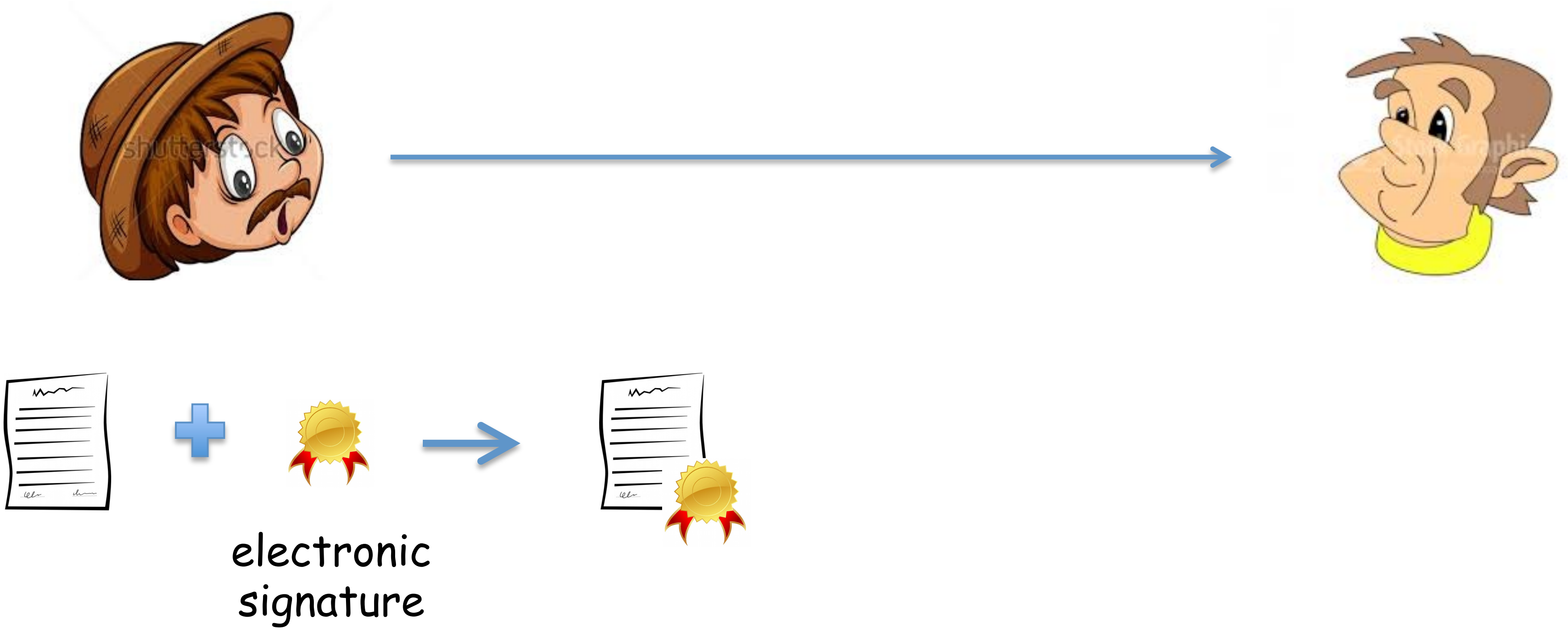
Digital Signature Scheme

signing electronically



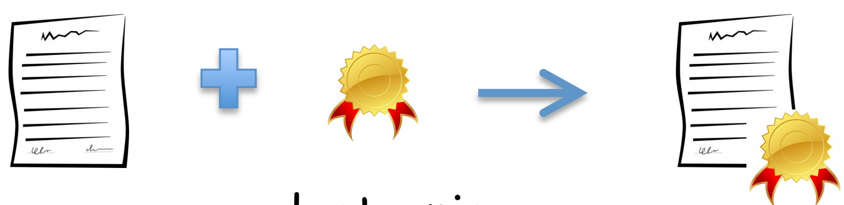
Digital Signature Scheme

signing electronically



Digital Signature Scheme

signing electronically

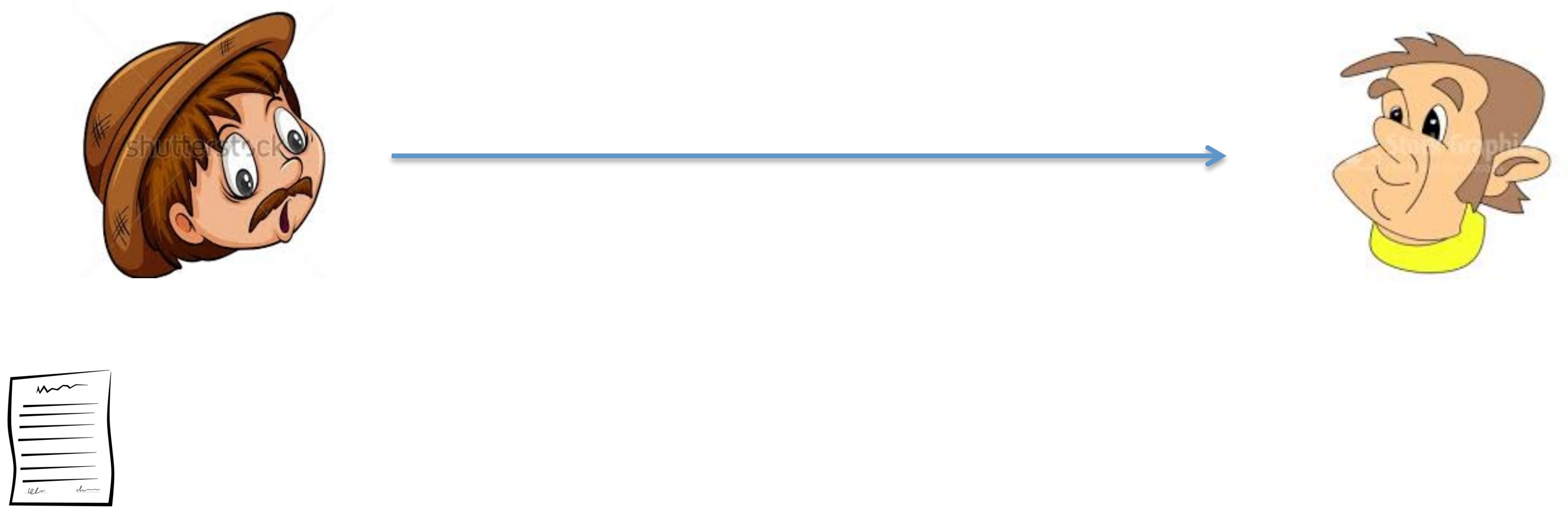


electronic
signature

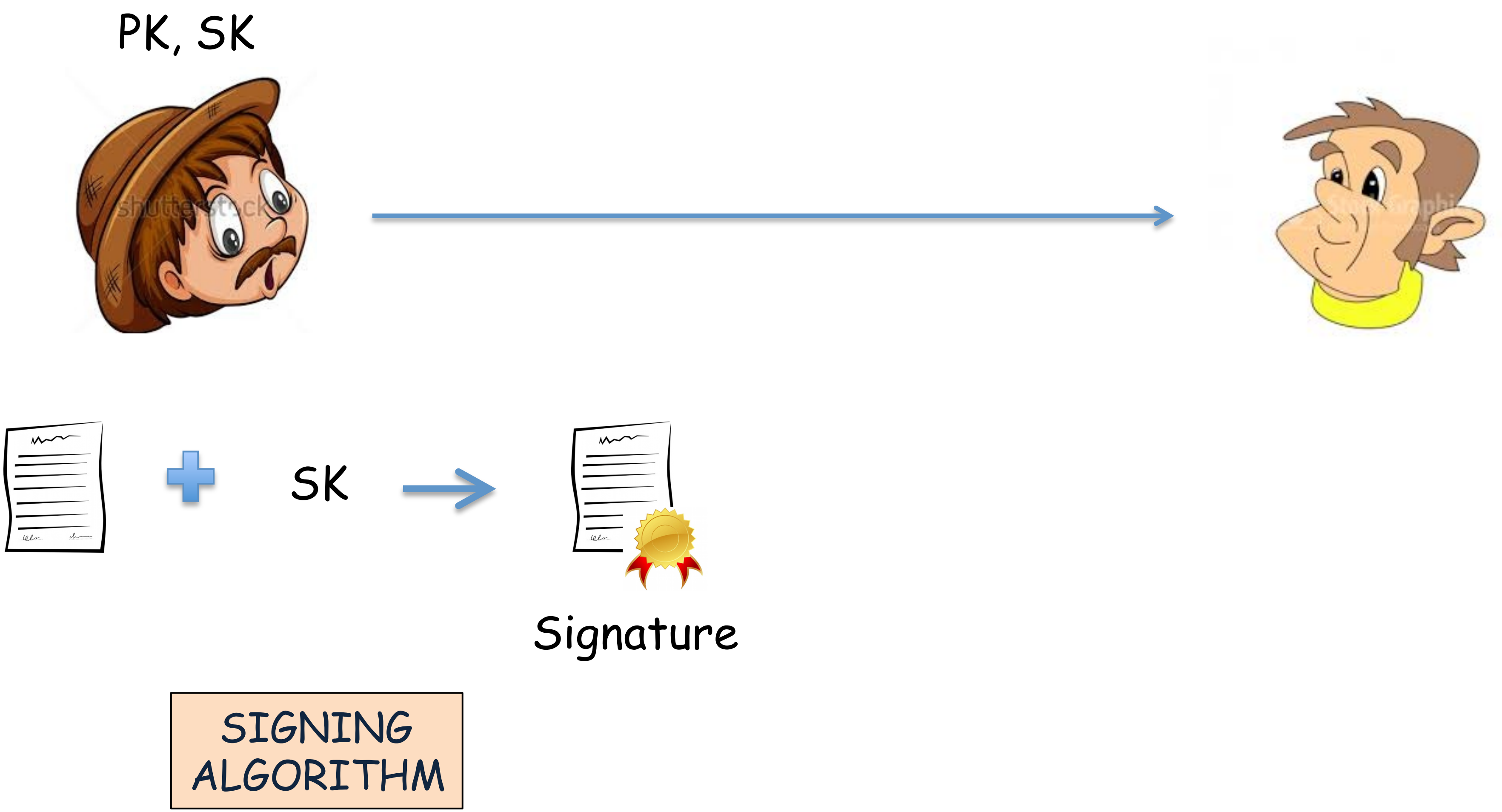
- signature can be easily copied
- it should be a function of the message

Digital Signature Scheme

PK, SK

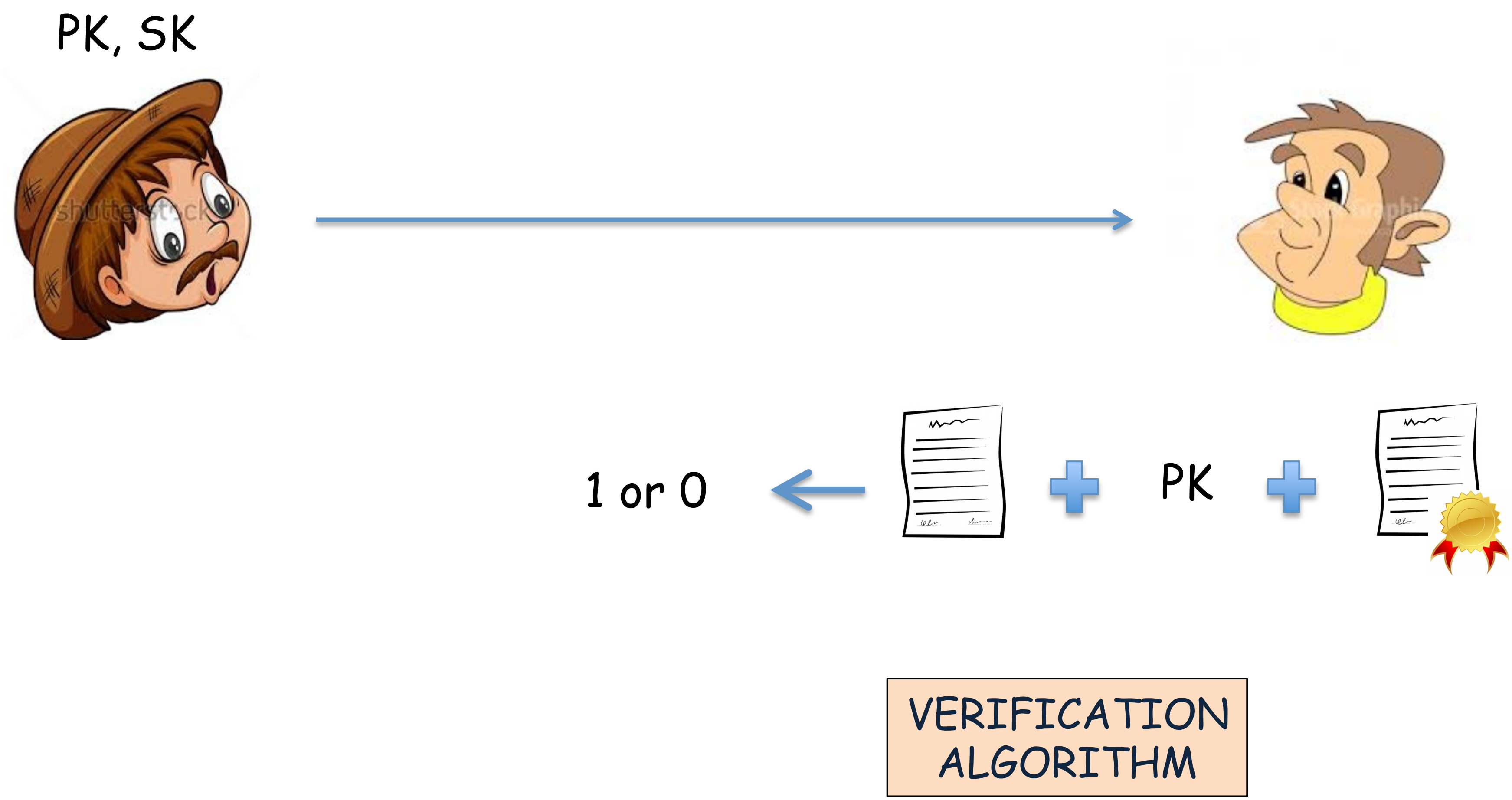


Digital Signature Scheme



Digital Signature Scheme

Digital Signature Scheme



Digital Signature Scheme

A digital signature scheme consists of three algorithms

Gen : outputs a key pair (pk, sk)

Sign :takes a message m in M and the signing key sk as inputs
and outputs a signature σ on m

Verify : takes a signature σ, the public key pk, and a message
m as inputs and outputs 1 or 0

Correctness

For all (pk, sk) output by Gen and for all m in M

$$\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$$

Digital Signature Scheme

A digital signature scheme consists of three algorithms

Gen : outputs a key pair (pk, sk)

Sign :takes a message m in M and the signing key sk as inputs and outputs a signature σ on m

Verify : takes a signature σ, the public key pk, and a message m as inputs and outputs 1 or 0

- Integrity
- Authenticity
- Non-repudiation

Correctness

For all (pk, sk) output by Gen and for a

$$\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1$$

RSA Signature

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)

RSA Signature

PK=(N,e)

SK=(N, d)

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)
- keep (N, d) as secret key, and publish (N, e) as public key

SK=(

1024-bit prime p (309 digits) :

14926660406676521425746589984505259593698043308528112047243863356010910984506208081319
56748971365259498401849653125052988699487229776494690230843615504129894860602079175805
40454081140587353862234445577520476872543676486167892443872308705026778461121261224322
495328346630383486386663628878772838449087770123303

1024-bit prime q (309 digits) :

11613613323752462862307997343676166615781213580255442213388439971627821582770818854043
09941587431632243604740043902608510350793965690708054362041417166453772064699311683053
51122258807934047024235765278566582937247825531441295648260124631056178986340098086793
666788683120626019654875802245983332214723863553333

2048-bit N = p*q (617 digits) :

17335246217810680499565282364130282347913694411139706552337646969996795185310539972695
21358952194887788871014810831418332247519311546653852372027284816592666735822538434338
92884640596924123847468319293906862022798176422316189203111527718629657728492287223809
26373552800043250590230507345247504584516585217552163181827225685419709962073929610117
85207875481813218795712875845153649877824714771313687872723823283851257056268551307467
39659929219301975845456600691347974780165760856198842806361918614258903112139836688041
47423192923778212236303196414996652277121672303217925415867248268691221399027188630076
689585126618899

e= 65537

d=156889308164396431400692065987710672887346835620783590145841716088838352812353946228
91972529034215081468682805358018205828859592737807144047750439346373279344813134904276
35456103680016686842205912449803850909739266099781320495323886360922086895776441251828
09693149600565934559448607974452395629861213088638273908322193647517562366779545756246
94333459199323797014292572744820690951743368632774427103258270737146365143542038629603
86987521680125465264397787114761980772967265876932453895158342739562236679770844723218
94791941724936692758146268591864077998906212085463254632733257316467651948214324936532
49128075792100409

RSA Signature

PK=(N,e)

SK=(N, d)

RSA Signature

PK=(N,e)

SK=(N, d)

Signing

$\sigma = m^d \pmod N$ where m in $(Z_N)^*$

RSA Signature

PK=(N,e)

m, σ →

SK=(N, d)

RSA Signature

PK=(N,e)

SK=(N, d)

Verification

if $m = \sigma^e \pmod{N}$, then output 1;

otherwise, output 0

RSA-FDH

PK=(N, H, e)

SK=(N, H, d)

KeyGen

- pick two large primes p and q
- compute N = p.q
- choose an exponent e such that gcd(e,phi(N)) = 1
- choose an exponent d such that e.d = 1 mod phi(N)
- choose a function $H : \{0,1\}^* \rightarrow Z_N^*$
- keep (N, H, d) as secret key, and publish (N, H, e) as public key

RSA-FDH

PK=(N, H, e)

SK=(N, H, d)

Signing

$\sigma = H(m)^d \pmod{N}$ where m in $\{0,1\}^*$

RSA-FDH

PK=(N, H, e)

σ

SK=(N, H, d)

RSA-FDH

PK=(N, H, e)

SK=(N, H, d)

Verification

if $H(m) = \sigma^e \pmod{N}$, then output 1;

otherwise, output 0